DOCUMENT RESUME

ED 207 584 IR 009 702

AUTHOR Miller, Mark L.; Goldstein, Ira P.
TITLE PAZATN: A Linguistic Approach to Automatic Analysis of Elementary Programming Protocols. AI Memo 388.
INSTITUTION Massachusetts Inst. of Tech., Cambridge. Artificial Intelligence Lab.
SPONS AGENCY National Science Foundation, Washington, D.C.
REPORT NO LOGO-35
PUB DATE Dec 76
GRANT NSF-EC40708X
NOTE 94p.

EDRS PRICE MF01/PC04 Plus Postage.
DESCRIPTORS Classification; Computer Assisted Instruction; Computer Graphics; *Computer Programs; Flow Charts; Linguistic Theory; *Models; *Problem Solving; *Programing
IDENTIFIERS LOGO System; *Structured Planning and Debugging Editor; *Tutorial Mode

ABSTRACT

PATN is a design for a machine problem solver which uses an augmented transition network (ATN) to represent planning knowledge. In order to explore PATN's potential as a theory of human problem solving, a linguistic approach to protocol analysis is presented. An interpretation of a protocol is taken to be a parse tree supplemented by semantic and pragmatic annotation attached to various nodes. This paradigm has implications for constructing a cognitive model of the individual and designing computerized tutors. Manual protocol analysis is tedious and informal; hence the design for PAZATN, an automatic protocol analyzer, is presented. PAZATN uses PATN as a generator for possible interpretations of the protocol, with bottom-up evidence biasing PATN toward plans which are likely to match the data. PAZATN is a domain independent framework for constructing specialized protocol analyzers. To apply PAZATN to a particular task domain, event specialists (ESP's) are needed which embody syntactically organized domain knowledge. ESP's for the LOGO graphics programming domain are defined and PAZATN's operation is hand-simulated on an elementary protocol for this domain. Forty-one references are listed. (Author/LLS)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U S DEPARTMENT OF HEALTH
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

ED207584

IR009702

Massachusetts Institute of Technology

Artificial Intelligence Laboratory

AI Memo 388 December 1976 Logo Memo 35

PAZATN: A Linguistic Approach to Automatic Analysis of Elementary Programming Protocols

Mark L. Miller and Ira P. Goldstein

PATN is a design for a machine problem solver which uses an augmented transition network (ATN) to represent planning knowledge. In order to explore PATN's potential as a theory of *human* problem solving, a linguistic approach to protocol analysis is presented. An *interpretation* of a protocol is taken to be a *parse tree* supplemented by semantic and pragmatic annotation attached to various nodes. This paradigm has implications for constructing a cognitive model of the individual and designing computerized tutors.

Manual protocol analysis is tedious and informal; hence the design for PAZATN, an automatic protocol analyzer, is presented. PAZATN uses PATN as a generator for possible interpretations of the protocol, with bottom-up evidence biasing PATN toward plans which are likely to match the data.

PAZATN is a domain independent framework for constructing specialized protocol analyzers. To apply PAZATN to a particular task domain, *event specialists* (ESP's) are needed which embody syntactically organized domain knowledge. ESP's for the Logo graphics programming domain are defined and PAZATN's operation is hand-simulated on an elementary protocol for this domain.

This is a revised version of a report describing research supported in part by the Intelligent Instructional Systems Group at Bolt Beranek and Newman, under contract number MDA 903-76-C-0108 jointly sponsored by Advanced Research Projects Agency, Air Force Human Resources Laboratory, Army Research Institute, and Naval Personnel Research & Development Center. The research was conducted at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Investigation of the application to elementary programming was supported in part by the National Science Foundation under grant C40708X, and in part by the Division for Study and Research in Education, Massachusetts Institute of Technology.

Contents

Book I: A Linguistic Approach to Protocol Analysis

1. Introduction
2. An Example of Protocol Analysis as Parsing
3. Toward a Cognitive Model of the Individual
4. Notes to Book I

Book II: Automating the Protocol Parsing Process

5. Introduction to Book II
6. Simulating Automatic Parsing of the Example Protocol
7. Organization of the PAZATN Protocol Parser
8. Refining the Protocol Parser
9. Conclusion
10. Notes to Book II

References

Thanks are due to J. Aiello, J.S. Brown, S. Purcell, and S. Rosenberg for carefully criticizing preliminary drafts of this report. The authors would also like to thank Carol Roberts for assistance with the illustrations.

Book I: A Linguistic Approach to Protocol Analysis

1. Introduction

1.1. SPADE: A Linguistic Theory of Design
1.2. PATN: Analysis by Synthesis
1.3. Theoretical Interpretations

1.1. SPADE: A Linguistic Theory of Design

In recent research we have developed a theory of design called SPADE which provides a model of the planning and debugging processes.[1] We contend that, in addition to being a powerful theory of machine problem solving, SPADE is also a useful framework for describing human problem solving. To support this contention, we apply the SPADE theory to the task of analyzing problem solving protocols.

By adopting this methodology we follow the precedent established in seminal protocol analysis studies conducted at Carnegie Mellon University [Newell 1966; Newell & Simon 1972; Waterman & Newell 1972, 1973; Bhaskar & Simon 1976]. Our work extends their approach along three dimensions.

1. With the exception of the recent Bhaskar & Simon effort, the CMU studies have been restricted to very limited domains such as cryptarithmetic. Rather than limiting the task domain, we limit the range of responses. Typically protocols are transcriptions of think-aloud verbalizations; we focus on the more restricted interactions arising from a problem solving session at a computer console.[2] The analysis task in this setting is to interpret user actions -- editing, executing, tracing, etc. -- in terms of the SPADE theory of planning and debugging.

2. The CMU theory centers on the *production systems model*. Although productions are Turing universal, they tend to result in a less structured program organization than the linguistic formalisms of the SPADE theory. The PATN program, the procedural embodiment of the SPADE theory, uses an *augmented transition network* [Woods 1970] to represent planning knowledge.

3. CMU analyses are based on the *problem behavior graph*. Pursuing an analogy to computational linguistics, we define an interpretation of a protocol to be a *parse tree* supplemented by semantic and pragmatic annotation. The parse tree characterizes the constituent structure of the protocol. Semantic and pragmatic annotation -- variables and assertions attached to nodes of the parse tree -- formalize the problem description and the rationale for particular planning choices. Annotated parse trees closely reflect the local structure of PATN's linguistic problem solving machinery, leading more directly to inferences regarding individual differences than is evident from problem behavior graphs.

Ruven Brooks [1975] applied the CMU approach to the programming domain, developing a model of *coding* -- the translation of high level plans into the statements of a particular programming language -- and testing the model by analyzing protocols. His model is a set of production rules whose conditions match the patterns of plan elements and whose actions generate code statements. Protocols are analyzed manually, with the experimenter attempting to infer the plan which is then expanded by the production system into code paralleling that of the protocol. The processes of understanding the problem, generating the plan, and debugging are not formalized. SPADE goes beyond this in that it can be used to parse protocols and that the parse constitutes a formal hypothesis regarding not only the coding knowledge but also the planning and debugging strategies employed by the problem solver.

The paper is divided into two books. Book I develops SPADE's linguistic paradigm for protocol analysis. A prototypical elementary programming protocol is parsed, and the implications of this information processing analysis for constructing cognitive models and designing computerized tutors are discussed.

Book I does not address the question of how a protocol parse is derived. In earlier work, problem solving protocols were analyzed manually.[3] However, manual analysis is tedious and informal; hence Book II presents the design for PAZATN, an automatic protocol analyzer. PAZATN uses PATN[4] -- the procedural

embodiment of the SPADE theory -- as a generator for possible interpretations of the protocol, with bottom-up evidence biasing PATN toward plans which are likely to match the data (figure I:1).

PAZATN is a domain independent framework for constructing specialized protocol analyzers. To apply PAZATN to a particular task domain, *event specialists* (ESP's) are supplied which embody domain-specific knowledge. For concreteness, we employ examples from the Logo elementary graphics programming domain; ESP's for this domain are discussed. PAZATN's operation is hand-simulated on an elementary protocol from this domain.

1.2. PATN: Analysis by Synthesis

A major insight of the generative grammarians (e.g., Chomsky [1965]) was that it is often helpful to characterize phenomena *synthetically*: one devises rules to *generate* the phenomena. Analysis can then be viewed as a recognition process for selecting derivations from the space of synthetic possibilities. We adopt this viewpoint in analyzing protocols, with PATN as our generative formalism.

The SPADE theory, which PATN embodies, begins with a taxonomy of commonly observed planning techniques (figure I:2). When a problem is confronted -- according to the theory -- one of three types of plans may be pursued. *(1)* The problem may be solved by *identification*: recognizing it as already having a solution. This planning category, seemingly trivial, is of course essential to avoid infinite regress. *(2)* The problem may be solved by *decomposition*: dividing it into smaller, easier subproblems. These are each solved separately, and then recombined, thereby disposing of the original problem. *(3)* Should the first two strategies fail, the problem may be solved by *reformulation*: redescribing it in terms that seem more amenable to solution. The reformulated

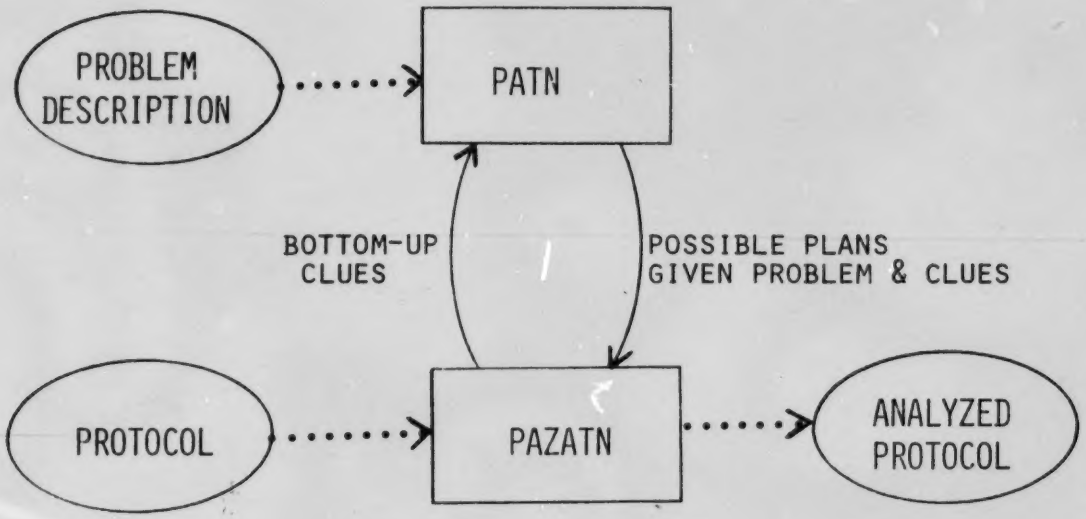


FIGURE I:1 TOP LEVEL ORGANIZATION OF THE PROTOCOL ANALYZER

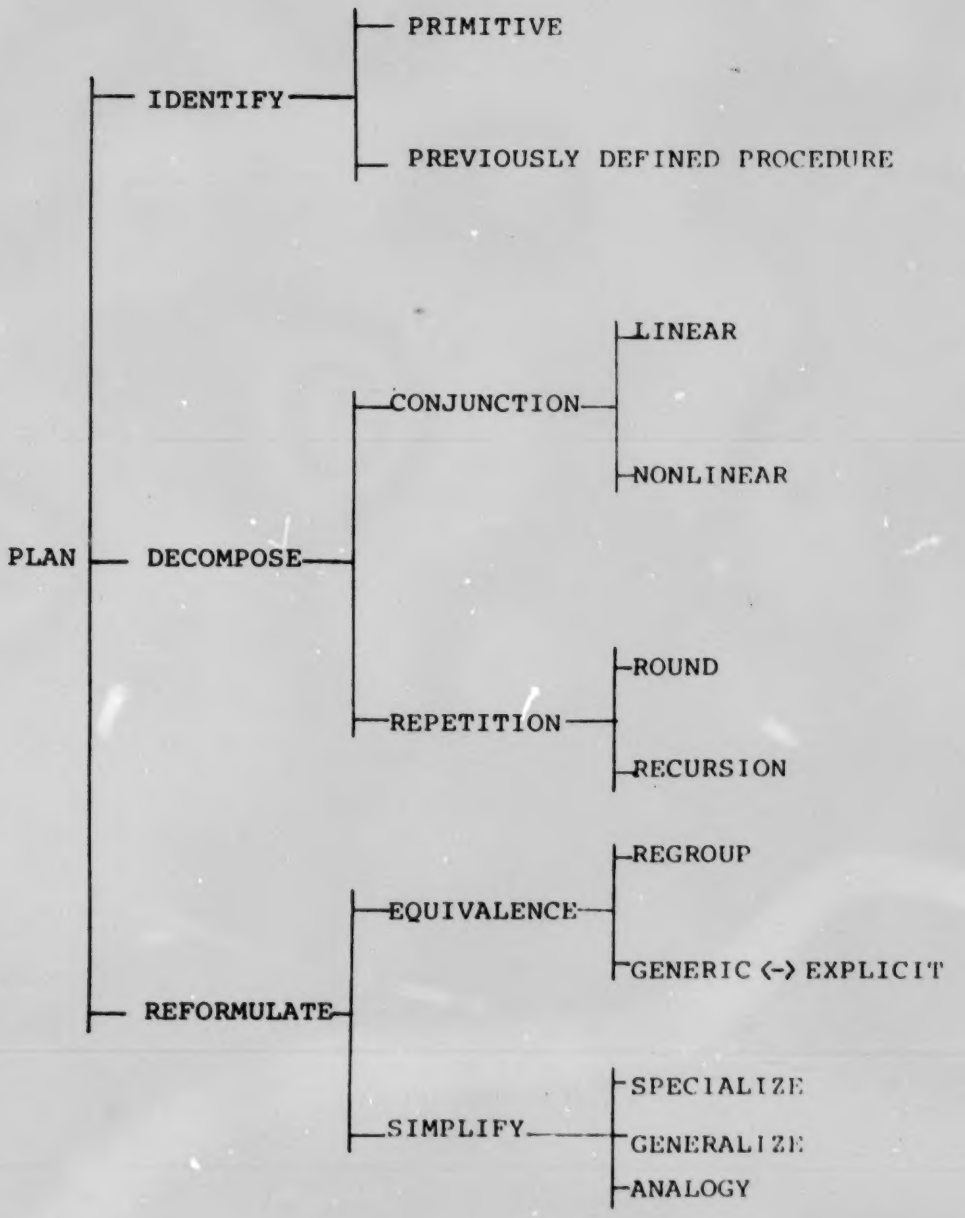


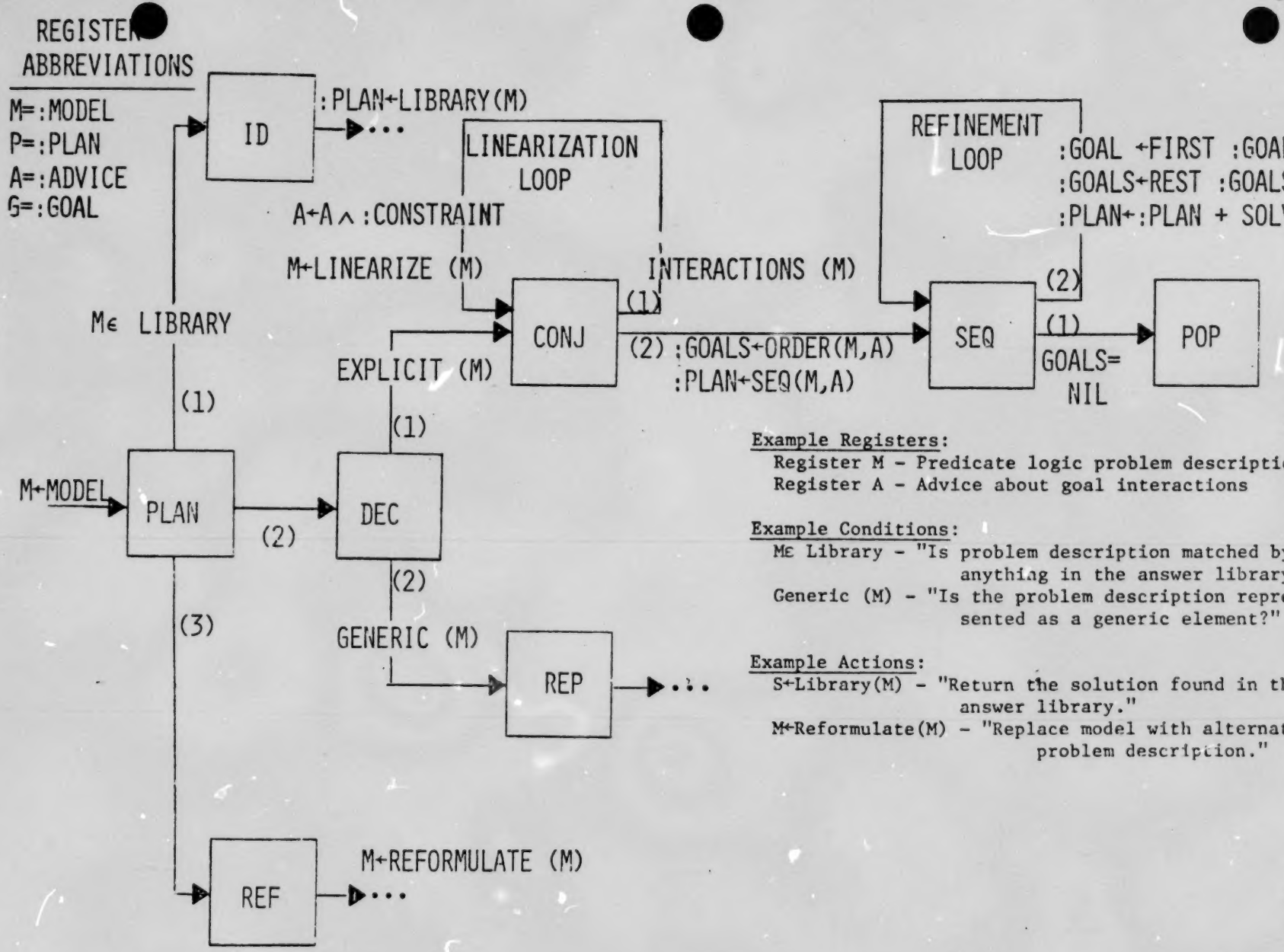
FIGURE I:2
TAXONOMY OF PLANNING CONCEPTS

problem must, in turn, be solved by identification, decomposition, or further reformulation. As the figure indicates, each of these categories of plans is further subdivided.

PATN (figure I:3) is a problem solving program based on this taxonomy. PATN was derived by first representing the taxonomy as a recursive transition network.[5] This produces a non-deterministic problem solver. Supplying precedence ordering for arcs from each node, predicates which test preconditions for transitions, and actions to be performed when arc transitions occur, produces an augmented transition network [Woods 1970] that is far more deterministic in solving problems (although backup is permitted). The predicates access registers which store semantic information about the problem; the actions modify these registers.

PATN's solutions can exhibit *rational bugs* -- errors arising from heuristically justifiable but incorrect planning decisions -- such as the trial execution of an *incomplete plan*, which omits necessary interface steps. Hence a complementary theory of debugging is developed using the same approach as in the planning theory. Figure I:4 shows a taxonomy of debugging techniques. This taxonomy bifurcates into techniques for *diagnosing* the underlying cause of a bug and techniques for *repairing* the bug once isolated. *Model diagnosis* is typical of the diagnostic techniques. It consists of executing the program in order to construct a list of violated model predicates, which is then examined to check if any code was written to accomplish the violated predicates. As in the planning theory, the debugging taxonomy is transformed into an ATN (figure I:5) by providing registers, arc predicates and so on. The debugging ATN is called DAPR (*d*ebugger of *a*nnotated *pr*ograms), and is an integral part of the PATN system.

Consider the operation of the planning ATN on an example from the Logo graphics environment, where students define, test, and debug procedures to draw



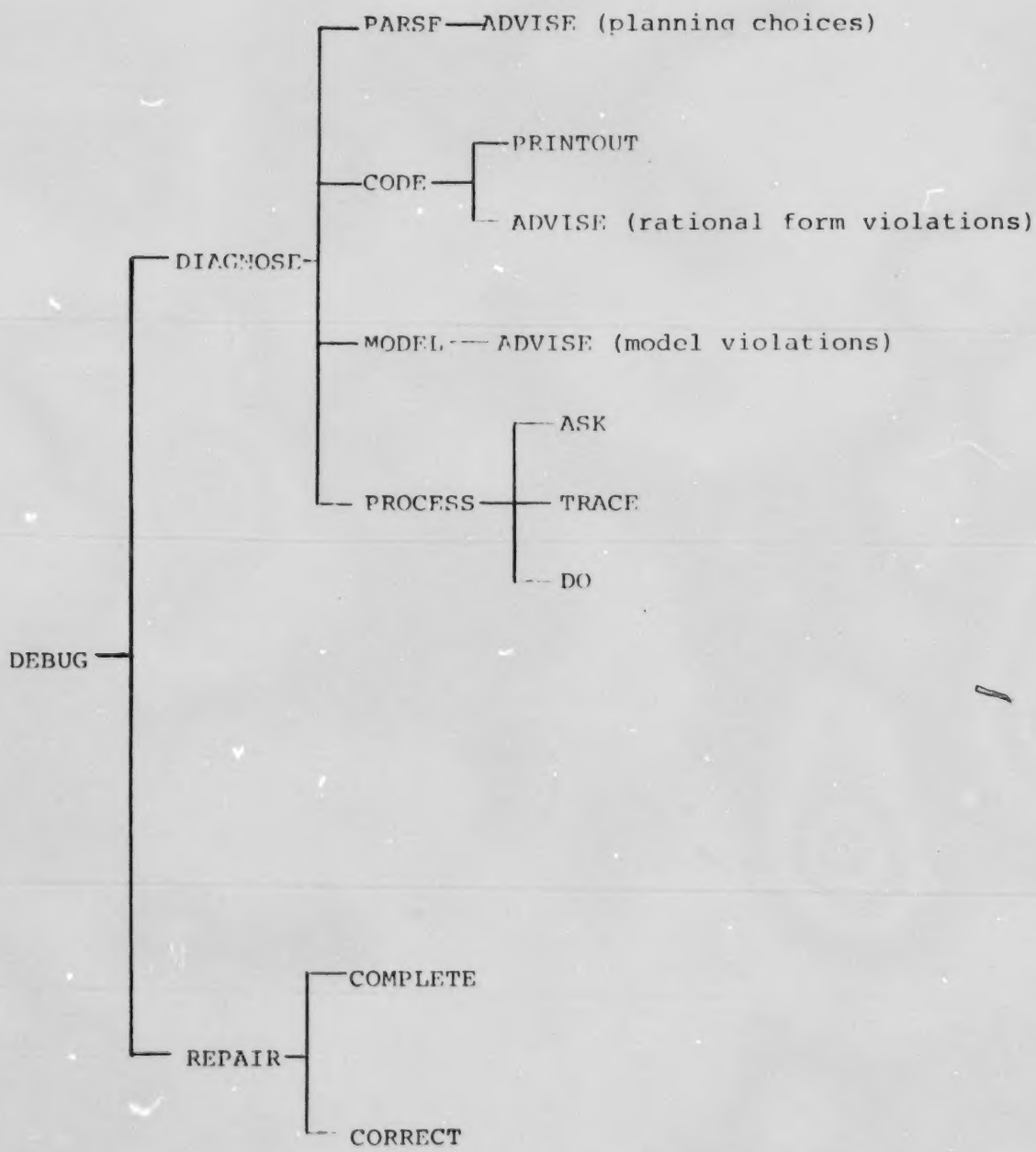
Example Registers:
Register M - Predicate logic problem description
Register A - Advice about goal interactions

Example Conditions:
Mε Library - "Is problem description matched by anything in the answer library?"
Generic (M) - "Is the problem description represented as a generic element?"

Example Actions:
S←Library(M) - "Return the solution found in the answer library."
M←Reformulate(M) - "Replace model with alternative problem description."

FIGURE I:3 A SIMPLIFIED VIEW OF PATN

FIGURE I:4 A TAXONOMY OF DEBUGGING TECHNIQUES



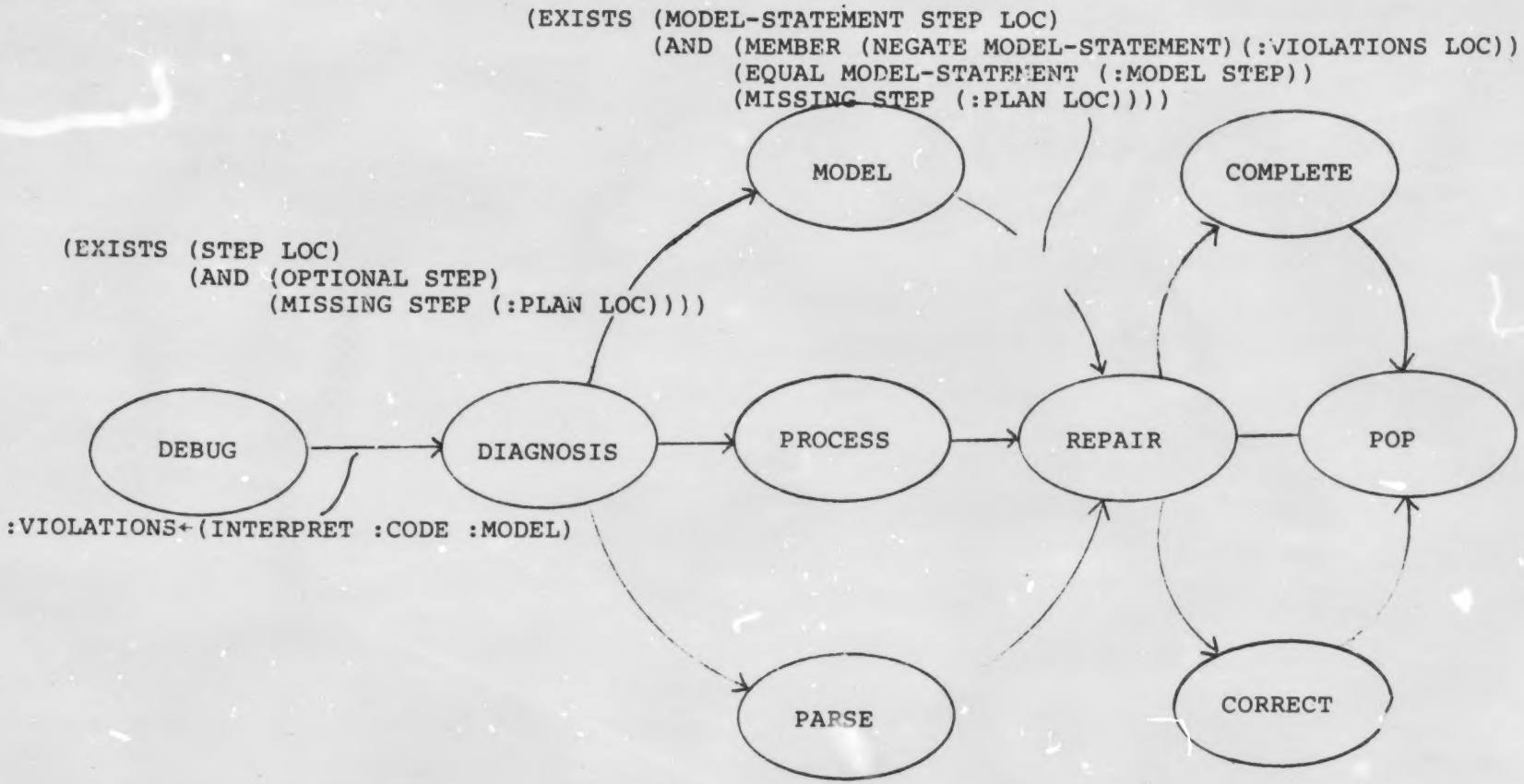


FIGURE I:5 DAPR: PATN'S DEBUGGING ATN

simple pictures. The WISHINGWELL (figure I:6) is a typical beginner's project. The students' task description is a sketch of the desired picture. PATN, however, requires a formal task description: figure I:7 illustrates this description, the *model*, for WISHINGWELL. Models are expressed in an assertional formalism developed by Goldstein [1974, 1975], which is similar to the first order predicate calculus. The model characterizes the range of pictures which match the sketch.[6]

PATN's solution to the WISHINGWELL task has three aspects: *(1)* an hierarchical plan derivation, summarizing the arc transitions which were followed; *(2)* a snapshot of the values of the ATN's registers attached to each node of the derivation, representing the semantic context at the time the node was created; and *(3)* a set of instantiated arc predicates at each node describing why the chosen arc transition was preferred to its competitors; these are called the pragmatic assertions of the node.[7] The semantic variables and pragmatic assertions relate the subgoal structure of the problem solving protocol to the model describing the task to be accomplished.

Figure I:8 shows PATN's hierarchically annotated solution. Naturally, this is not the only solution to the WISHINGWELL problem: to apply PATN to protocol analysis, we allow PAZATN to reject solutions which do not match the protocol data, forcing PATN to backup so that alternative solutions are generated.

1.3. Theoretical Interpretations

We define an *interpretation* of a protocol to be a PATN plan derivation: a parse tree whose fringe is the list of events (e.g., figure 1:8), augmented by annotation associated with each node of the parse. Since different plans sometimes lead to the same coding events, some protocols have more than a single

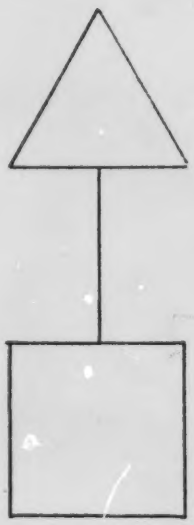


FIGURE I:6
WISHINGWELL PICTURE
AN ELEMENTARY LOGO GRAPHICS PROJECT

Draw a WISHINGWELL with a square well and a triangular roof, connected by a pole which is a line. The roof should be above the pole, and the pole above the well. The well should be connected to the pole at the midpoint of the upper side of the well and the lower endpoint of the pole. The pole should be connected to the roof at the midpoint of the bottom side of the roof and the upper endpoint of the pole. The bottom side of the roof and the upper side of the well should be horizontal.

```
(DEFINE-MODEL WISHINGWELL ()
   (EXISTS (ROOF POLE WELL)
           (AND (TRIANGLE ROOF)
                (LINE POLE)
                (SQUARE WELL)
                (ABOVE ROOF POLE)
                (ABOVE POLE WELL)
                (EXISTS (P)
                        (AND (CONNECTED WELL POLE (AT P))
                             (EQ P (MIDDLE (UPPER (SIDE WELL))))
                             (EQ P (BOTTOM (ENDPOINT POLE)))))
                (EXISTS (Q)
                        (AND (CONNECTED POLE ROOF (AT Q))
                             (EQ Q (MIDDLE (BOTTOM (SIDE ROOF))))
                             (EQ Q (UPPER (ENDPOINT POLE)))))
                (HORIZONTAL (BOTTOM (SIDE ROOF)))
                (HORIZONTAL (UPPER (SIDE WELL))))))
```

Figure 1:7. Predicate Model for a Wishingwell

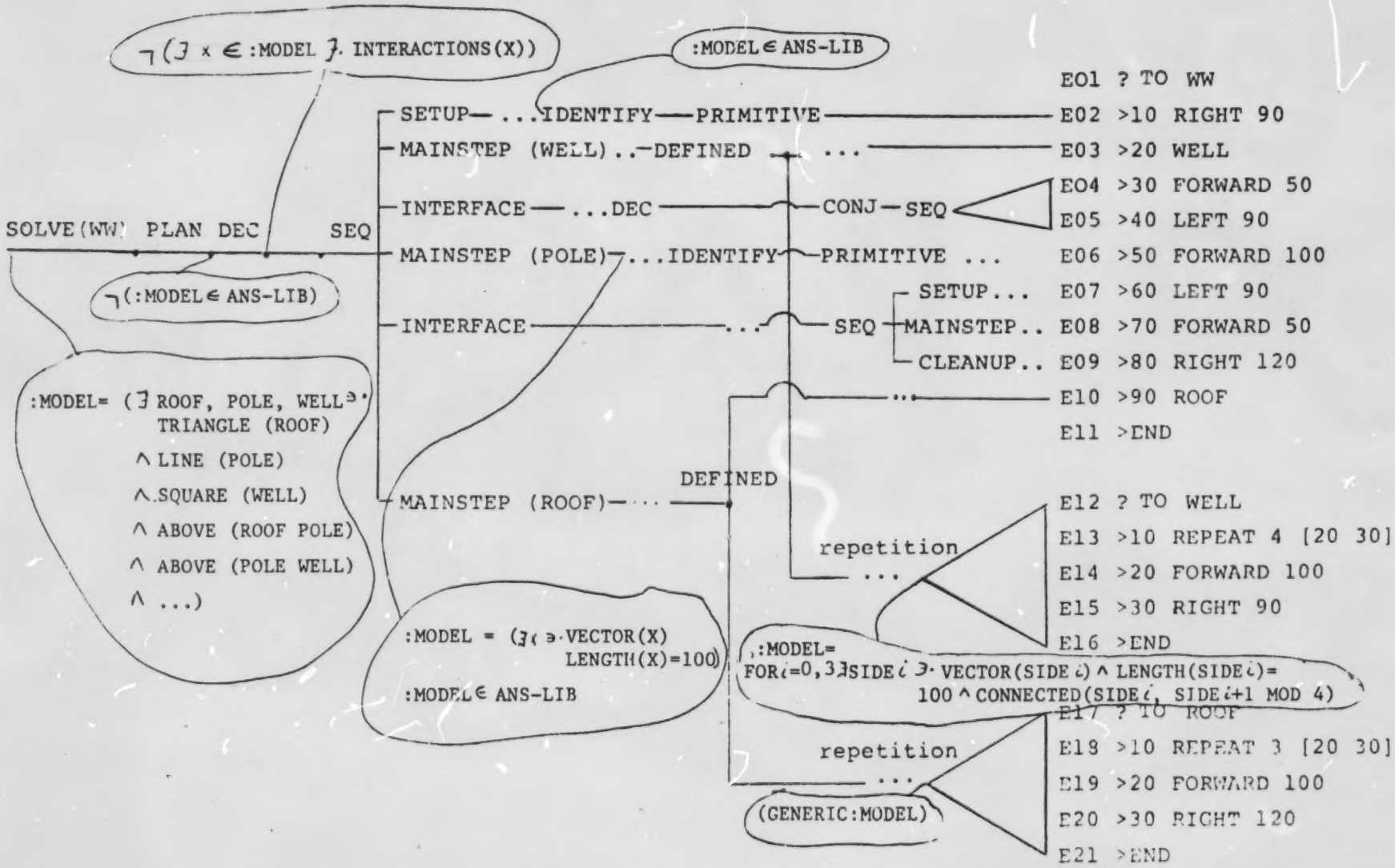


FIGURE I:8 PATN'S ANNOTATED DERIVATION TREE FOR THE WISHINGWELL TASK

interpretation. The basic claim of this paper is that PATN can efficiently generate the psychologically plausible interpretations.

Evidence for this claim rests on four sources of evidence.

1. The heuristic adequacy of PATN as a problem solving program provides suggestive, though by no means decisive, evidence. At least for the restricted world of elementary Logo graphics, hand-simulations indicate that PATN is heuristically adequate.

2. Introspection by human problem solvers is a weak but useful source of evidence. To some extent PATN was designed on the basis of introspection and hence has some support along this dimension.

3. A strong source of evidence is the appropriateness of the replies of a question-answering module that performs retrievals and simple inferences over a database composed of these interpretations. The question-answering module is introduced in chapter two. The replies to the example questions given in that chapter seem appropriate to the authors.

4. The strongest source of evidence is ability to predict performance in future situations on the basis of past behavior. Chapter three describes modifications to the ATN that provide predictive models of typical problem solving behaviors.

We find this informal evidence sufficiently encouraging (as detailed in the remainder of Book I) to warrant the design (in Book II) of a precise framework for generating SPADE-style protocol interpretations. Future research will rigorously evaluate the psychological validity of these interpretations as follows.

1. PATN will be implemented and tested on a broad range of examples. This will confirm its heuristic adequacy.

2. An editor based on SPADE will be constructed as a structured programming environment, and transcripts of the problem solving behavior of programmers using this editor analyzed. Coupled with systematic interviews, this will provide evidence regarding the sufficiency of SPADE's repertoire of planning and debugging concepts.

3. PAZATN with a question-answering interface will be implemented. The appropriateness of the replies generated by the question-answering module will be judged by skilled but unbiased informants, and by systematic subject interviews.

4. A modeling component will be implemented that modifies the PATN ATN to be more in accord with a particular student's behavior. Tests will be conducted to determine whether the modified ATN is more successful in predicting performance on subsequent protocols.

Before proceeding, a possible misconception involving the distinction between representational frameworks and psychological theories should be dispelled. Two hypotheses are defended by the research program outlined in this paper: *(1)* that ATN's are a *useful representation* for the models we are developing; and *(2)* that *particular* ATN's, the output of our modeling procedure, constitute *theories* of *individuals* -- stated in the language of ATN's -- which make statements about the presence or absence of certain problem solving skills. Both hypotheses are of course subject to experimental verification. We do not argue that other Turing-universal formalisms (such as productions or Heidorn's [1975] augmented phrase structure grammars) cannot also represent these theories. Stronger claims regarding the *validity* of ATN's *per se* as psychological mechanisms require additional assumptions regarding processing costs and limitations which we are not currently prepared to defend.

2. An Example of Protocol Analysis as Parsing

2.1. An Example Problem Solving Protocol
2.2. Structural Description
2.3. Semantics
2.4. Pragmatics

An analogy to computational linguistics has been fruitful both in defining the objectives of analysis and in designing the PAZATN system for automating the analysis process. The analogy suggests partitioning analyses into syntactic, semantic, and pragmatic components. These components correspond to the potential control paths, data flow, and branching conditions of a procedural problem solver. From a problem solving standpoint, these are modeled by the network of states and arcs, the registers, and the transition conditions of the augmented transition network. From a protocol analysis standpoint, syntax is represented as a parse tree; semantics and pragmatics are represented as annotation (variables and assertions) associated with each node of the parse tree.

This chapter presents a SPADE interpretation for a typical WISHINGWELL protocol. Book II provides a hand-simulation of PAZATN generating this interpretation.

2.1. An Example Problem Solving Protocol

Since analysis consists of the selection of a PATN plan derivation, analyzing a protocol identical to PATN's default solution (figure 1:8) is trivial. Hence, a different protocol, involving a variety of plans including reformulation and repetition, serves as our example.[8]

The student begins by writing an iterative procedure to draw the square WELL.

```
E01     ?TO WELL
E02     >10 REPEAT 4 [20 30]
E03     >20 FORWARD 100
E04     >30 RIGHT 90
E05     >END
```

A superprocedure for the WISHINGWELL is defined by a sequential plan drawing first TREE, a previously defined procedure, and then WELL.

```
E06     ?TO WW
E07     >10 TREE
E08     >20 WELL
E09     >END
```

The WW program is executed, producing figure I:9.

```
E10     ?WW
```

The program is edited to include an interface establishing the proper relation between TREE and WELL.

```
E11     ?EDIT WW
E12     >13 RIGHT 90
E13     >15 FORWARD 50
E14     >17 RIGHT 180
E15     >END
```

2.2. Structural Description

The result of analyzing this protocol is a data structure, the interpretation, consisting of syntactic, semantic, and pragmatic components. The syntactic component, diagrammed in figure I:10, is the protocol's *structural description*: a parse tree representing the sequence of PATN arc transitions required to generate it.[9]

Such structural descriptions capture one aspect of problem solving behavior. They provide a formal basis for answering questions regarding which plan types were used, a topic which could otherwise be discussed only intuitively.[10] Their most direct application is to answering "how questions."

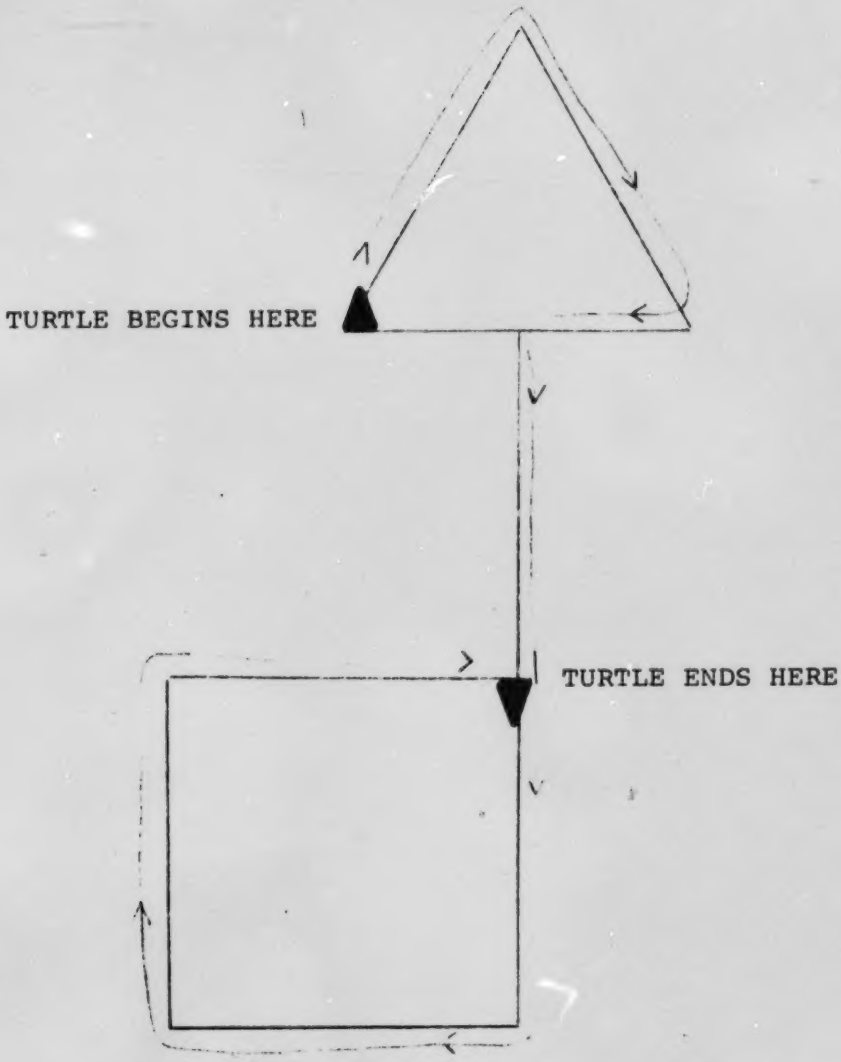


FIGURE I:9 WW AT E10 -- INTERFACE NEEDED

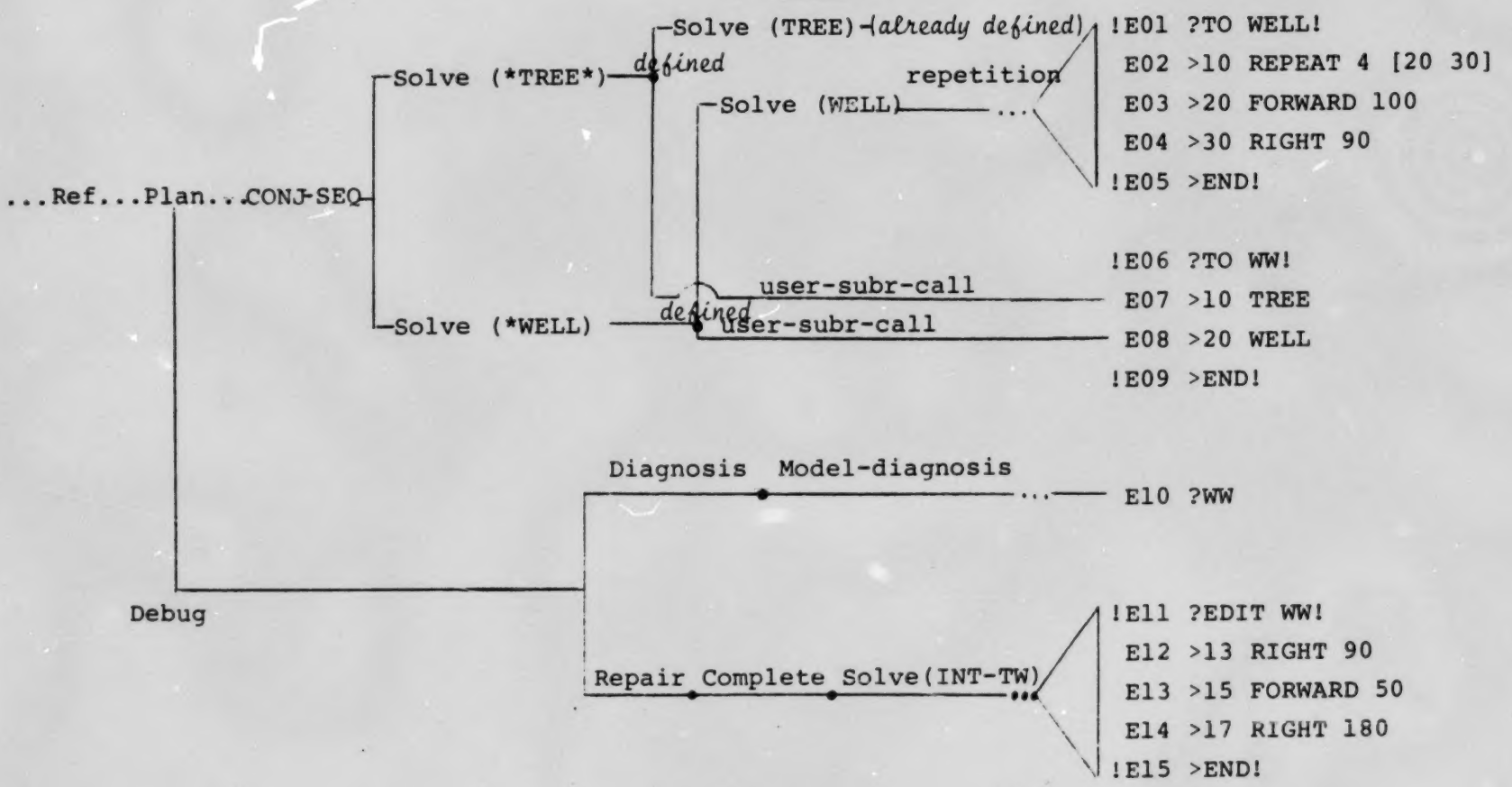


FIGURE I:10 ABBREVIATED STRUCTURAL DESCRIPTION FOR STUDENT WW

Q1. How does procedure WELL accomplish a square?

A1. WELL uses a repetition plan. The generic subgoal is a side.

Q2. How does procedure TREE accomplish a roof and pole?

A2. The roof and pole model parts were regrouped into a tree by a reformulation plan. Procedure TREE was already in the answer library, allowing an identification plan.

Still, the parse tree is an incomplete description: it does not indicate the semantic relationships between subgoals or the pragmatic criteria governing the choice of one plan over another.

2.3. Semantics

Semantic annotation is defined to be the values of semantic variables associated with each node of the parse. These variables relate the plan to the formal problem model by recording the contents of the ATN's registers at the time the node was generated. The following are typical PATN registers.

1. :PLAN is the hierarchically annotated program defined below the current node, reflecting its state after dominated editing events have been processed.
2. :CODE is the fringe of :PLAN.
3. :EFFECT is a description of the effect of executing the code defined below the current node. Since the code may contain references to undefined user procedures, :EFFECT may be unassigned at a given node. For the elementary graphics domain, this variable is called :PICTURE, and describes the picture drawn by the program in Cartesian coordinates.
4. :MODEL is the set of predicates which :CODE is intended to accomplish. For a correct program :EFFECT is an instance of :MODEL.
5. :ADVICE is a list of planning suggestions generated by PATN arc actions. For example, the linearization arc (see PATN's conjunction node in figure 1:3) creates advice regarding both the order in which subprocedures should be written, and the order in which they should be invoked.

6. :CAVEATS is a list of warnings for potential bugs generated by PATN when heuristic guidelines are used in planning. For example, if no interactions are detected when solving a problem involving an unfamiliar domain predicate, it is possible that the predicate actually give rises to interactions, but their patterns have not yet been learned by the system. Hence, :CAVEATS can be set, recording this potential bug, on the arc transition from conjunction to sequential plans. This information guides DAPR, PATN's debugging module, in subsequent diagnosis.

7. :VIOLATIONS is the list of model predicates which are not satisfied by the :EFFECT achieved by :CODE. This register and :EFFECT are set by a *performance annotation* module designed by Goldstein [1974].

Let us sample the values of the semantic variables at various nodes of the parsed WW protocol. :MODEL for the top level SOLVE node was shown in figure 1:7. For the INT-TW SOLVE node, :MODEL is:

> *The tree must be above the well, and the bottom endpoint of the tree must connect to the midpoint of the upper side of the well.*

In our LISP-oriented model language notation this is represented as:

```
(AND (ABOVE TREE WELL)
     (EXISTS (P)
             (AND (CONNECTED TREE WELL (AT P))
                  (EQ P (MIDDLE (UPPER (SIDE WELL))))
                  (EQ P (BOTTOM (ENDPOINT TREE)))))).
```

This submodel reflects the reformulation of WISHINGWELL into a TREE and a WELL.

Typically, semantic annotation is relevant to answering "what questions." The above value of :MODEL for the INT-TW node provides an example.

Q3. What is the purpose of lines 13, 15 and 17 of WW?

A3. Those three lines are in-line code interfacing subprocedures TREE and WELL. The interface establishes connectivity at the appropriate point, and causes the tree to appear above the well.

:VIOLATIONS at the PLAN node for WW provides another example.

Q4. What is wrong with procedure WW when it is first executed (at event E10)?

A4. The necessary relations between the model parts TREE and WELL have not been established: specifically, there is no point P such that the tree is connected to the well at P, P is the middle upper side of the well, and P is the lower endpoint of the tree.

The :VIOLATIONS variable which mediates this answer is non-empty at the PLAN node because the debugging which generates the missing interface has not yet occurred: the English answer simply paraphrases its LISP value:

```
(NOT (EXISTS (P)
             (AND (CONNECTED WELL TREE (AT P))
                  (EQ P (MIDDLE (UPPER (SIDE WELL))))
                  (EQ P (BOTTOM (ENDPOINT TREE)))))).
```

2.4. Pragmatics

Pragmatic annotation is defined to be a record of the justifications for selecting a given arc transition over its competitors, and constitutes an hypothesis about the reasons for using a particular plan. REASONs are assertions attached to each node of the parse. *The REASON for using a particular plan in a particular situation is an instance of the arc predicate leading to the ATN state for that plan, where the current values of the registers are taken into account.*[11] For example, the reason that WELL was decomposed using a repetition plan in the protocol is that :MODEL at that node was generic.

```
(REASON (REPETITION E02)
        (GENERIC (:MODEL E02))).
```

Pragmatic annotation is germane to answering "why questions."

Q5. Why did the student execute WW at event E10 -- did (s)he believe the program to be correct?

A5. Probably the student expected bugs. A reasonable strategy is to initially plan only for the main steps, with the interfaces solved later by debugging. WW was executed at E10 in order to discover what interfacing, if any, was needed.

This illustrates the analysis of a procedure containing the rational bug of constructing an incomplete plan.[12] Debugging operations are analyzed by postulating the application of some DAPR technique. The reasons for debugging operations typically involve localizing or repairing the cause of some model violation. The purpose of running the program at E10 was to perform model diagnosis; this technique was chosen because the occurrence of two consecutive mainsteps (with no explicit interface) implies that the plan may be incomplete:

```
(REASON (MODEL-DIAGNOSIS E10)
        (AND (OPTIONAL (INTERFACE TREE WELL))
             (MISSING (INTERFACE TREE WELL)
                      (:PLAN E06)))).
```

In this case, model diagnosis demonstrates the existence of violated predicates for which no code exists: the plan is in fact incomplete. This is the reason for the subsequent editing: repair of the incomplete plan by resuming planning at the offending locale.

The reason for the completion plan in the editing episode (E12 through E14) is to eliminate the violations by supplying the missing interface between TREE and WELL.

```
(REASON (COMPLETE (E12 E13 E14))
        (AND
         (MEMBER
             '(NOT
               (EXISTS (P)
                       (AND (CONNECTED WELL POLE (AT P))
                            -- )))
             (:VIOLATIONS E10))
         (EQUAL
             '(EXISTS (P)
                      (AND (CONNECTED WELL POLE (AT P))
                           -- ))
             (:MODEL (INTERFACE TREE WELL)))
         (MISSING (INTERFACE TREE WELL) (:PLAN E10)))).
```

The conjunction of predicates collectively called SEQ (on the arc from conjunction to sequential) plays a role in the following example.

Q6. Why was the invocation order (TREE WELL) used, rather than the reverse?

A6. TREE ends at its bottom, a required connection point, resulting in simpler interfacing for that ordering. If the TREE *began* at that connection point, the reverse order would have been preferable.

Here one of the SEQ predicates incorporates knowledge about the domain predicate CONNECTED, that interfacing can be simplified if two subprocedures are invoked in an order such that the endpoint of the first corresponds to a mutual connection point. An instance of this rule becomes a pragmatic assertion of the SEQ node in the parse.

The reason for preferring the (TREE WELL) sequencing is that TREE ends at a required connection point of WELL.

```
(REASON (SEQ (E07 E08))
        (AND
         (EQ (POSITION :TURTLE (AFTER TREE))
             (BOTTOM (ENDPOINT TREE)))
         (EQ (POSITION :TURTLE (AFTER TREE))
             (MIDDLE (UPPER (SIDE WELL))))))
```

A precise definition of a linguistic approach to protocol analysis has been provided and a concrete analysis of this kind supplied. We now turn our attention to the potential utility of the approach for constructing cognitive models of individuals.

3. Toward a Cognitive Model of the Individual

3.1. Tailoring the ATN to the Individual
3.2. Individual Differences and Overlay Modeling
3.3. Issues and Examples, and the Computer as Coach

3.1. Tailoring the ATN to the Individual

Advocates of computer-aided instruction point out that computers can be used to tailor instruction to the needs of the individual. Yet little is known about what it means to construct cognitive models of individual students, or about how to use them in providing sensitive and effective automatic tutoring. The SPADE theory suggests an approach.

SPADE confronts the problem of individual differences by considering the possible ways in which the student's ATN can differ from that of an expert. One error would be to have a variant of the optimal pragmatic arc constraints. More serious would be to have missing or extra arcs. Even more serious would be to have missing or extra states. Differences which can be formalized as alterations to the topology of the ATN are manifested in the production of a different set of parse trees: PATN might be capable of some derivations not available to the student, or vice versa. Differences in arc conditions or arc actions are manifested by the selection of other than the optimal plan for a particular problem situation, although the same repertoire of plans may be available.

These types of modifications, properly combined, can account for many commonly observed weaknesses in student problem solving. To demonstrate this point, we present six examples of student weaknesses and the fashion in which our modeling scheme is able to capture them. The examples are derived from informal data collected in our prior Logo tutoring experiences.

1. *BASIC Syndrome*: A student with prior programming experience in the BASIC language never uses recursion. Problems for which iteration is awkward are solved only with difficulty; problems for which iteration is inadequate, such as drawing arbitrarily deep binary trees, are unsolvable.

A deviant version of the PATN subgraph for repetition planning is illustrated in figure I:11. The correct subgraph has an intermediate ROUND plan state; the deviant version, missing this state and its associated arcs, characterizes the BASIC syndrome. The student's repetition arc bypasses the ROUND state, *short circuiting* the ATN to pursue the iteration option with no possibility of recursion. In general, failure to employ a full repertoire of planning options can be modeled in this fashion: the short circuit is postulated to occur at the node immediately prior to the least common superset of the class of unused plans.

2. *Discontinuity*: A student fails to build upon previous work, never taking advantage of relevant existing procedures. Each new picture to be drawn is treated as an isolated problem, and recurring subproblems are repeatedly solved afresh.

PATN can accomplish identifications using either primitives or previously solved problems. Discontinuity amounts to examining the primitive library only. This is modeled by the absence of the corresponding predicate on the arc from PLAN to IDENTIFY. A similar but more subtle case would be a student that *occasionally* uses previous solutions, but not as often as PATN predicts. This indicates that the identification network is probably intact, but parts of the reformulation subgraph are missing. Such a student fails to notice the relevance of previous problems because they are described in slightly different terms. Introspection suggests that this is a common source of difficulty.

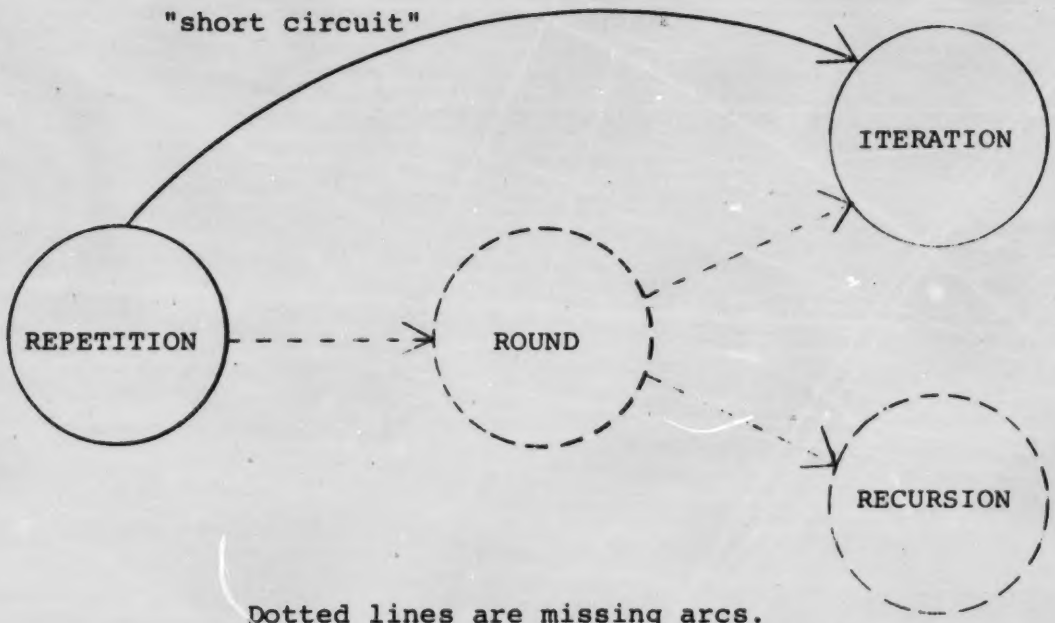


FIGURE I:11 DEVIANT ATN SUBGRAPH FOR REPETITION PLANS

3. *Diagnosis Avoidance*: A student performs well in planning and defining programs. However, when a bug occurs, the student falters. Rather than systematically localizing the underlying cause of the error, followed by repair, the student immediately begins to edit the program. The changes are haphazard and counterproductive, creating more bugs than they eliminate.

Relative to the DAPR debugging ATN, diagnosis avoidance is a weakness wherein the student has an extra arc not present in the expert. Whereas DAPR cannot proceed to the REPAIR state without first passing through DIAGNOSIS, the student is modeled as having an undesired extra arc bypassing this state (figure I:12). This allows diagnosis to be (incorrectly) treated as optional.

4. *Syntactically Unstructured Code*: A student never uses subprocedures, instead relying entirely on in-line code. This results in long, unreadable programs which are difficult to debug. Often the student forgets which subgoals have been solved, or forgets how previously solved code segments work. Few projects are successfully completed.

PATN's use of subprocedures is governed by register setting actions associated with the sequential refinement loop. This is the culpable locale for a type of non-modular design we call *syntactically unstructured code*. Instead of first setting the :PLAN register to a sequence of subprocedure calls, and then pushing for a solution to each in turn, the student apparently performs these actions in the reverse order: first pushing for a solution to each subprocedure, and then setting the :PLAN register to the concatenation of the popped results. Note that this deviant ordering of arc actions requires far more intermediate storage to keep track of recursive calls to the ATN: given a limited pushdown stack, it is not surprising that the student forgets things.

5. *Semantically Unstructured Code*: A student mechanically begins every Logo procedure with the PENUP command. Usually this works out well, in preparation for a position setup. However, even when the position setup is unnecessary, the PENUP is still used, resulting in either: (a) a *rational form violation*, in which

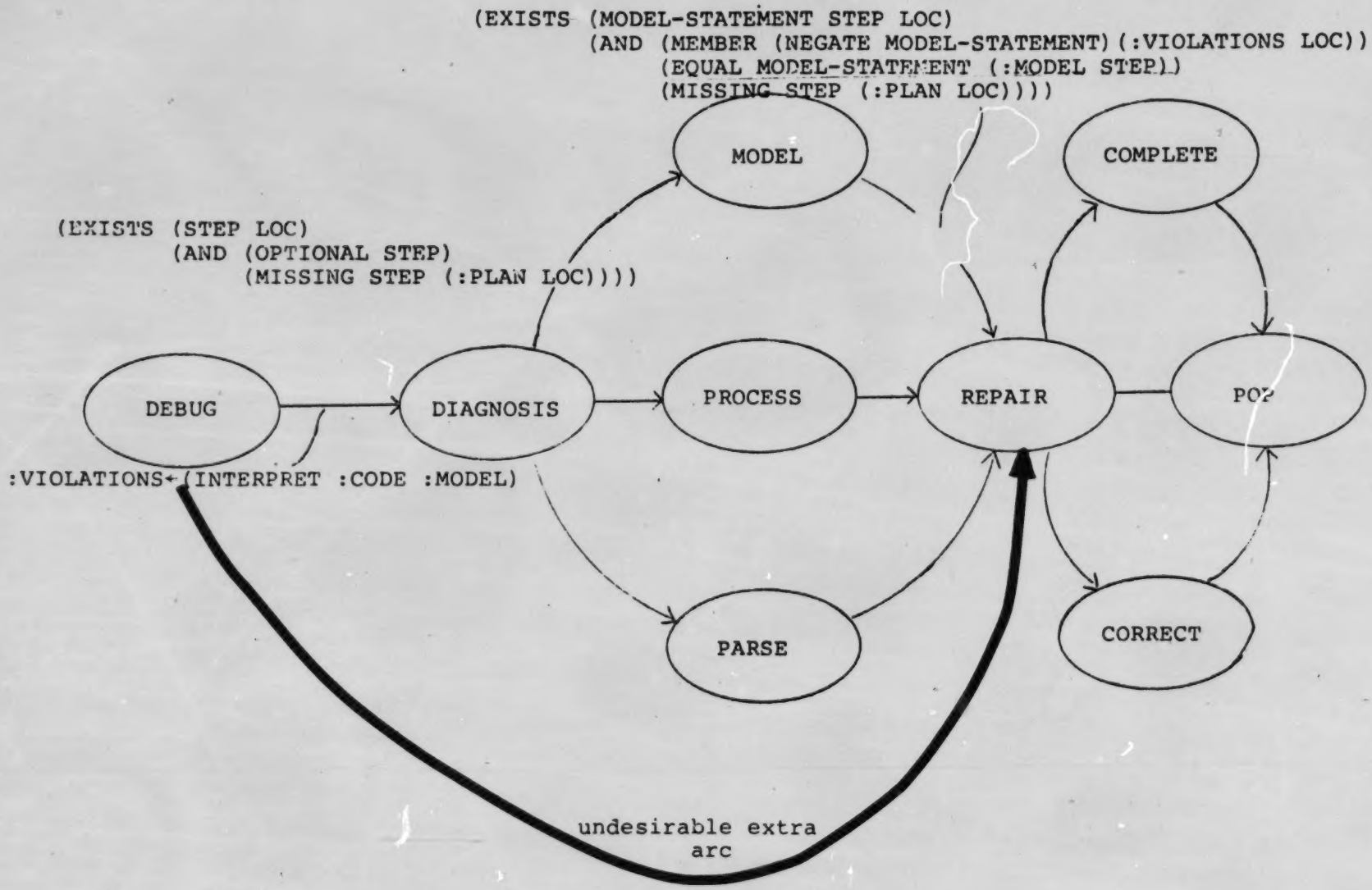


FIGURE I:12 DEVIANT ATN SUBGRAPH BYPASSING DIAGNOSIS

the PENUP is followed immediately by a PENDOWN; or (b) one or more missing model part violations, due to the invisibility of vectors intended to accomplish main steps. (Solomon [1976] uses the term *cliché* to describe this class of phenomena.)

Treating some optional constituent as if it is required results in a second kind of non-modularity, *semantically unstructured code*. The particular *cliché* just described, mechanically including PENUP commands, is mediated by the linear decomposition arc. If this arc is modified to create position setup subgoals without testing whether :MODEL actually requires such a setup, the effect is to include a PENUP at the start of each turtle program.

6. *Pragmatically Unstructured Code*: A student who normally *does* break large programs into subprocedures nevertheless encounters numerous bugs, many of which are difficult to localize. The subprocedures lack modularity, each being dependent on knowledge of the inner workings of others. For example, interfaces are included as part of main steps, so that the initial state of a given procedure is determined by the final state of whichever procedure happens to precede it in the planned order of invocation.

While failure of a particular arc action to consider the problem at hand results in semantically unstructured code, faulty arc predicates in deciding among alternative arcs leads to a third form of non-modularity, *pragmatically unstructured code*. The unnecessary construction of non-linear subprocedures is attributable to either improper default ordering or malfunctioning predicates on the arcs leaving the conjunction node. For example, the INTERACTIONS predicate may not be imposing sufficiently strong conditions on accepting the model: this leads to the addition of constraints on the subprocedures when no real non-linearity is present.

Thus perturbations on PATN provide a deep theory of student weaknesses, explaining unsuccessful behavior in terms of the syntactic, semantic, and

pragmatic structure of the ATN.

3.2. Individual Differences and Overlay Modeling

We envision inducing a model of individual idiosyncrasies as perturbations of PATN by applying Goldstein's [1976] *overlay modeling* technique. This approach describes individuals with respect to an expert problem solving program by associating probabilities with each decision point in the expert, representing our state of knowledge about a given individual's preferences. The probabilities are a summary of the available evidence rather than an integral part of the model: at any given time, a process model is obtained from the overlay probabilities by including those possibilities that are above threshold and excluding those that are below.[13] Goldstein and Carr [1977] use this technique to infer process models of behavior in a logic and probability game called WUMPUS.

This raises the question of whether all of the perturbations mentioned above, including the alterations in ATN topology, can in fact be represented by such an overlay, i.e., by a numerical *plausibility table*: it turns out that they can. A missing arc can be handled by assigning it an *a priori* transition plausibility of 0. A missing intermediate state can likewise be represented by the plausibility of the arcs leading to the unused states being 0, but the plausibility of the arc to the "short circuited" state being 1. Similarly, default orderings can be reversed by reversing their relative plausibilities.

This table driven organization allows distinguishing between *personal* and *archetypal* ATN's. Archetypal ATN's are analogous to Winston's [1970] concept models, and in fact our scheme for inducing personalized ATN's bears some resemblance to Winston's learning system, except that our networks happen to have procedural rather than structural meanings. Personalized ATN's are created from

the archetype by thresholding over a particular plausibility table. This simplifies the tuning and debugging of the system and eliminates the danger that states or arcs added to model non-expert behaviors might degrade the performance of the expert. The expert ATN is obtained by coupling the archetype with an expert plausibility table. The entries for undesirable options are zeroes.

A first approximation to the plausibility table for an individual can be derived from the relative frequencies of arc transitions in previously analzyed protocols. However, this ignores the semantic and pragmatic context. It could be that infrequently used transitions were inappropriate for the tasks performed. Consequently, this is refined by comparing the tallies to a record of the expert's performance over the same set of tasks. (This technique, *differential modeling*, is suggested by Burton & Brown [1976].) Naturally there will be differences in individual protocols because of arbitrary choices, but in the long term consistent properties of the student's behavior should emerge.

Just recording arc transitions is still too crude. One should account for differences in terms of the smallest chunks of malfunctioning knowledge which can be isolated. As a second order cognitive model, the units of analysis are taken to be the individual arc predicates and actions. The statistical evidence can be used to differentiate which arc operations are malfunctioning or missing.

3.3. Issues and Examples, and the Computer as Coach

Two crucial ingredients are lacking in current uses of computers in education: a cognitive theory describing the problem solving and learning processes, and a pedagogical theory prescribing techniques to facilitate and enhance these processes. As a result, many instructional applications of computers are *ad hoc*, if not detrimental.

There are exceptions to these criticisms. The Logo project [Papert 1971]

offers educational applications of computer technology suggested by a computational approach to problem solving and learning. However, the justification for many Logo insights remains informal and intuitive. The current work is an effort to increase the theoretical precision and experimental rigor of Logo research. Other exceptions include the work of John Seely Brown's group [Brown et al. 1974,1975; Burton & Brown 1976] on intelligent instructional systems for electronics (SOPHIE) and elementary mathematics (WEST), and that of Stansfield, Carr and Goldstein [Stansfield et al. 1976; Goldstein & Carr 1977] on an advisor for WUMPUS. The WEST tutor suggests a paradigm, also used in WUMPUS, in which *issues* (abstracted differences between expert and novice behavior) are illustrated by concrete examples of their application to active learning situations.

Given the cognitive modeling tools developed in this chapter, an issues-and-examples Logo tutor can be contemplated. When PATN's expectations are violated because of a difference between the expert and student versions of the ATN, then that issue can be raised with the student. This would extend the *issues-and-examples* paradigm of WEST and the *computer-as-coach* paradigm of WUMPUS, not only by addressing a more difficult task domain, but also by elaborating the notion of *issues*, from abstractions of empirically selected features, to specific programmatic weaknesses.

The theory would also constrain the order in which issues should be presented to the student. The topology of the ATN should be nearly right before pragmatic arc constraints are discussed. Likewise, the general form of the pragmatics should be correct before domain-specific *arc critics* are taught. Although many subtleties arise which are not touched on here, the approach takes a step toward theoretical foundations for computer tutors which provide sensitive, flexible, *individual* instruction in problem solving skills.

4. Notes to Book I

1. *SPADE* is an acronym for *S*tructured *P*lanning *a*nd *De*bugging. See [Goldstein. & Miller 1976a,b; Miller & Goldstein 1976a,b,c].

2. More accurately, the session transcript is a *partial* protocol. Considerable leverage is obtained by assuming that the dialogue occurs within the confines of a small, well-defined response menu: natural language processing need not be attempted. We recognize that thorough protocol analysis includes parallel examination of the subject's utterances during the session, eye movement data, retrospective accounts, and so on. Although our sole objective here is analysis of the session transcript, we intend to corroborate our analyses using these other sorts of evidence.

3. Miller & Goldstein [1976b] used a context free problem solving grammar to extract the constituent structure of a student's Logo protocol. That paper did not develop the more thorough view of analysis we describe in Book I of the current report.

4. PATN is designed in [Goldstein & Miller 1976b]. *It has not yet been implemented.* The use of present tense throughout this document in describing both PATN and PAZATN is for readability only.

5. For efficiency, some states with similar topology are merged, and a few additional arcs are added to provide for such features as iterative control, when recursively invoking the complete system is unnecessary.

6. The figure adopts a parenthesized notation (which is formally equivalent to that used in our earlier papers) to emphasize that predicate models are just LISP S-expressions which can be evaluated.

At first these predicate models will be supplied by the experimenter. Eventually we plan to construct a module to induce the model from a hand-drawn tablet sketch. A significant undertaking itself, this would enhance the practicality of automatic protocol analysis in the graphics domain.

7. Generation of pragmatic assertions representing instances of arc predicates is an elaboration of the basic PATN design, not presented in [Goldstein & Miller 1976b]. These assertions, being directly computable by examining the ATN's arcs and the semantic variables, are synthetically redundant, but become important when analytic complexities such as irrational bugs and personalized ATN's are considered.

8. The example is a simplified hypothetical protocol not involving careless errors such as mistypings. In other respects, however, it is typical of student protocols for tasks similar to WISHINGWELL.

9. The root of the parse tree is shown at the left; the leaves are to the right. Some details are not shown: ellipses are indicated by three periods. For clarity, some semantic information is included parenthetically: SOLVE(WELL).

Logo punctuation events are of minor importance in the underlying plan. Although used as clues during parsing, they are not included in the structural description. In the figure they are shown enclosed in exclamation marks: !EOS >END!.

Since the order in which subgoals are solved need not mirror their execution order in the resulting plan, events need not occur in the parse in temporal order. In the figure the events are shown in temporal order, but lines are crossed.

10. To illustrate the insights gained from the analysis, we use a scenario for a question-answering module which performs retrievals and simple inferences over a database consisting of the analyzed protocol. We are confident that the data structures generated by our style of analysis are sufficient to support this type of interaction. However, we have not yet designed the question-answering module *per se*; instead, we have concentrated on isolating the relevant knowledge base. For readability, the questions and answers are stated here in unrestricted English; for ease of implementation, the actual system will be restricted to a formal query language.

11. It might seem that this definition of pragmatic annotation is inadequate for protocol analysis, since a student may select the right plan but for the wrong reason. The SPADE approach handles this circumstance by a separate mechanism, *personalized ATN's*, to be discussed shortly. For ease of presentation, the example uses the expert ATN as the basis for its REASON assertions.

12. Although PATN's default solution to the wishingwell task did not involve debugging, PATN is capable of rational bugs such as this particular incomplete plan. When solving novel tasks, it is sometimes more efficient to plan only for the main steps, with the interfaces being solved by subsequent debugging. During planning, PATN notes those points where the plan is incomplete; when a bug is encountered, this advice guides PATN's debugging module, DAPR.

Not all rational bugs are incomplete plans, and not all bugs are rational. Overlooking an interaction between subgoals is another type of rational bug. Mistypings and mispellings are typical *irrational* bugs; our approach to their analysis should be mentioned. The reason for such an event is assumed to be the same as the reason for the correct version of the event, but flagged by an additional assertion stating the nature of the mistake.

13. Of course, one can also use probabilities to model actual non-determinism in the subject's behavior, but we do not consider that possibility here.

Book II: Automating the Protocol Parsing Process

5. Introduction to Book II

5.1. The CMU Series of Analyzers
5.2. Overview of PAZATN

Book I developed the SPADE notion of protocol analysis as parsing, but did not indicate how parses are to be derived. Automating the analysis process is desirable, because manual analysis is informal, tedious, error prone, and not amenable to incorporation into computerized tutors. Hence, this second book presents the design for PAZATN, an automatic protocol analyzer based on the SPADE theory. As background for assessing the design of PAZATN, we first summarize the features and limitations of a series of automatic protocol analyzers developed at Carnegie-Mellon University.

5.1. The CMU Series of Analyzers

Much ground-breaking research in automatic protocol analysis has been performed at Carnegie-Mellon University. PAS-I [Waterman & Newell 1972], the first of three CMU systems, analyzes think-aloud protocols for cryptarithmetic. PAS-II [Waterman & Newell 1973] is an interactive version which makes fewer task-specific assumptions. SAPA [Bhaskar & Simon 1976] addresses the additional complexities of semantically rich task domains.

By focusing on the cryptarithmetic task, PAS-I obtains sufficient leverage to completely automate the analysis process. The input to PAS-I is a transcription of a tape recorded think-aloud protocol and its output is a *problem behavior graph*. PAS-I operates in four stages, the first two of which occur sequentially in time: linguistic analysis, semantic analysis, processing of

operator groups, and problem behavior graph generation. PAS-I does not attempt generality; for example, the linguistic analyzer employs a key word grammar oriented to cryptarithmetic. Similarly, *Process-Column* is a typical operator.

PAS-II reduces dependency on a single domain by requesting guidance from a human encoder. Task-specific knowledge is factored into a separate set of rules; the domain independent part of the system amounts to a command language or subroutine library to assist a human protocol-analyst. Moving from automatic to interactive analysis may seem counter to progress. However, this methodological contribution allows flexibility to incorporate the experimenter's insight, while still imposing discipline on the encoding process. We intend to construct an interactive analyzer as an intermediate milestone in implementing PAZATN.

SAPA, in cooperation with a human encoder, analyzes protocols in chemical engineering thermodynamics. By considering a domain rich in background knowledge, rather than puzzle problems such as cryptarithmetic, SAPA addresses a complex new facet of problem solving. However, SAPA is highly domain specific. For example, SAPA begins the analysis by asking for the form of the energy equation used by the subject. Thermodynamics problem solving is viewed as a variant of means-ends analysis in which the energy equation plays a predominate role.

When implemented, PAZATN will extend the automatic protocol analysis techniques developed at CMU by complementing their features and limitations. On one hand, a PAZATN shortcoming -- its restriction to a small menu of responses -- is addressed by the considerable effort CMU researchers have invested in natural language front-ends for protocol analysis. On the other hand, CMU has devoted less attention to the investigation of planning concepts, a limitation addressed by the SPADE theory. For example, the CMU theory does not provide a deep account

of the origins of planning errors in the PATN sense. Likewise, a practical limitation of the CMU analyzers has been task specificity. In designing PAZATN we have tried to minimize task specificity through modular design; this is made possible, in part, by the highly structured underlying SPADE theory. However, testing the generality of PAZATN by applying it to several domains remains a research goal.[1] Finally, the elementary programming world to which PAZATN is applied in this paper resembles thermodynamics in that background knowledge of the domain plays a significant role in solving problems.

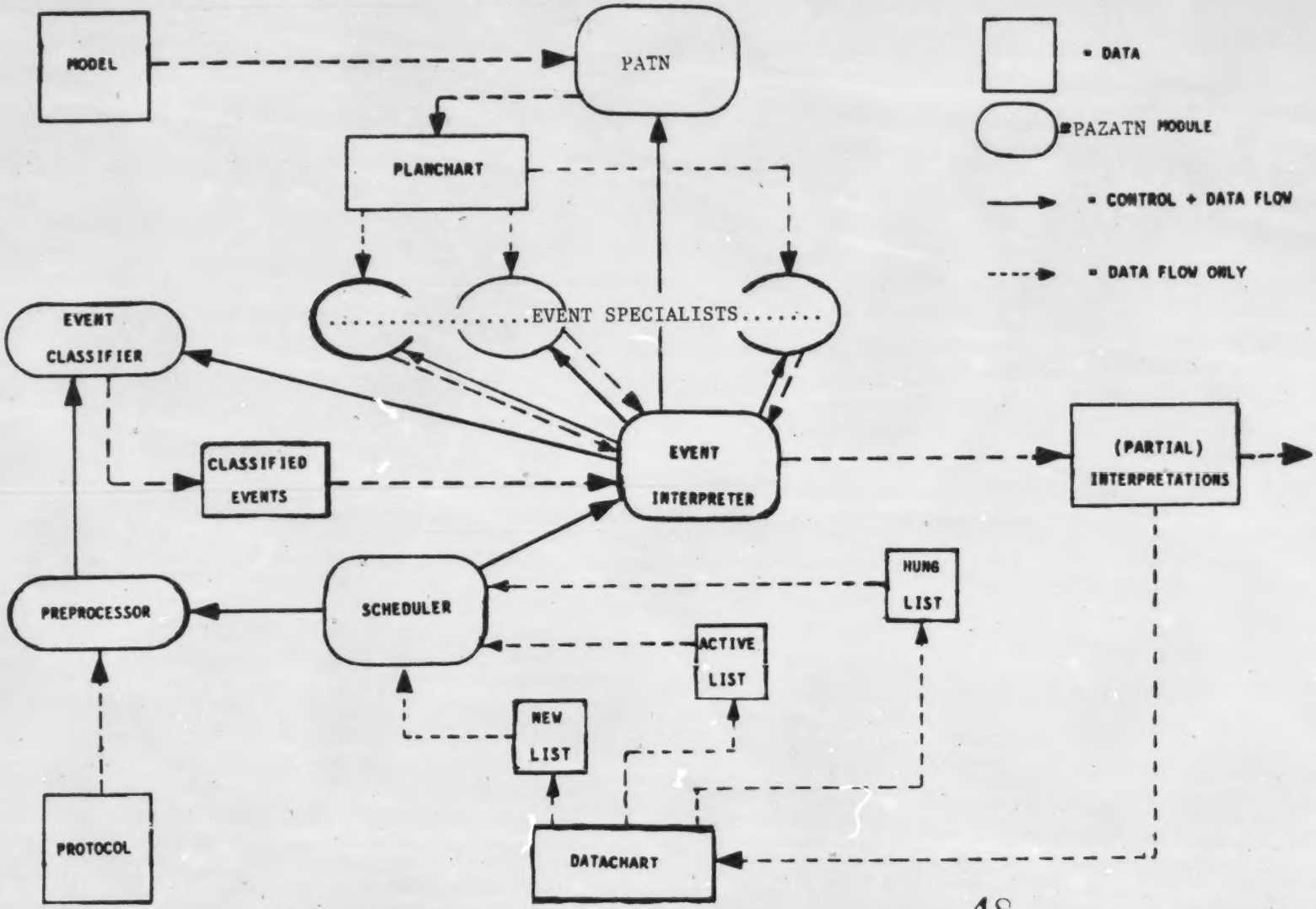
5.2. Overview of PAZATN

PAZATN is a scheme for matching a protocol to a PATN plan derivation; it can only understand protocols which PATN can generate.[2] Therefore the analysis could be performed, in principle, by trying all possible PATN solutions, selecting the first which matches the data. Since exhaustive enumeration is impractical, a primary consideration is efficient search in PATN's plan space. Bottom-up protocol evidence is used for this purpose (figure I:1).

PAZATN consists of PATN supplemented by several additional modules and data structures (figure II:1). This design incorporates three key ideas:

1. the use of the *chart data structure* [Kay 1973; Kaplan 1973] in two distinct roles, both involving the need to economically store alternative combinations of substructures;
2. the use of a *library of domain-specific specialists* for processing events in various syntactic categories;
3. the use of *best first coroutine search* driven by a separate scheduler -- with modules communicating by means of the charts -- to ensure early application of strong sources of constraint.

1. *Two charts:* One of PAZATN's charts, the *planchart*, keeps track of subgoals proposed by PATN. PAZATN's second chart, the *datachart*, records the alternative ways of associating protocol events with planchart leaves.



47

48

FIGURE II:1 BLOCK DIAGRAM OF PAZATN

2. *Library of event specialists:* The syntactic classification of possible events for a given domain results in a highly modular design. (ADDCODE, RUNCODE and END are typical Logo event types.) For each event type PAZATN is supplied with an ESP, i.e., a specialist for associating events of that type with planchart leaves. Adapting PAZATN to other problem domains is possible by replacing this library.

3. *Best first coroutine search:* PAZATN receives the model and protocol as input. The model is a formal statement of the problem as shown in figure I:7; the protocol is a list of events as shown at the top of page 1.19. PAZATN's output is a *SPADE interpretation* of the protocol as described in Book I: a parse tree augmented by semantic and pragmatic annotation. At any given time during analysis, several *partial* interpretations will be *active*. The outer loop is a scheduler which allows each active partial interpretation to examine one event per cycle. For a given interpretation, events are processed in a single left-to-right pass. At the end of a cycle the active set is re-chosen. This repeats until at least one interpretation has processed the final event.

Analysis of a protocol proceeds as follows. First PAZATN requests PATN to generate its most plausible plan on the basis of the model alone. This plan is inserted into the planchart. Next, protocol events are examined one by one, matching them with subgoals in the PATN plan. Each match is recorded in the datachart.

If an event is encountered for which no plausible match can be found, PATN is asked to generate its next most plausible plan, now potentially considering the nature of the mismatch as well as the model. The planchart is extended by inserting PATN's next plan. Those subgoals which are common to both plans share the same structure in the chart.

If an event is encountered for which more than one plausible match can be found, the datachart records each such pairing in similar fashion. Each of these is then allowed to continue examining protocol events according to the best first scheduling algorithm.

6. Simulating Automatic Parsing of the Example Protocol

6.1. Preliminary Generation of Expectations
6.2. Modifications Based on Bottom-up Evidence
6.3. Some Informal Observations

This chapter is a hand-simulation of PAZATN on the WISHINGWELL protocol introduced in Book I. Various components of PAZATN are introduced as they are needed. Subsequent chapters provide details regarding these components.

6.1. Preliminary Generation of Expectations

The protocol parsing process is initiated by executing *(PAZATN WISHINGWELL WW)*, where WISHINGWELL is the model and WW the protocol. The initial answer library is assumed to contain procedures for TRIANGLE and TREE.

Before PAZATN examines the protocol, PATN examines the model. Since WISHINGWELL is not in the answer library, PATN determines that an identification plan is not viable. Both decomposition and reformulation are possible, since they are applicable to any model.

PATN can determine, using lookahead, that reformulation results in an identification involving TREE; for this particular protocol, this quickly leads to a successful parse. However, reformulations rapidly expand the search space, so PAZATN adopts a conservative approach to reformulation: decompositions which lead to a straightforward solution are preferred unless protocol evidence indicating reformulation is discovered. Consequently decomposition is predicted, with three main steps: ROOF, POLE, and WELL. But since the decision is uncertain, a *demon procedure*[3] is created to handle the possibility that decomposition fails to parse the protocol.

The model is examined for interactions. None are detected, so a linear decomposition into subgoals is expected. However, since required connection

points occur at the midpoints of sides of WELL and ROOF, a non-linear subgoal decomposition might be used for efficiency, to avoid retracing. As a result, two demon procedures are created to check for WELL or ROOF sides being accomplished in two steps. Such a plan is less likely for ROOF which can be identified with the existing TRIANGLE.

The transitive ABOVE predicates suggest a sequential plan utilizing either the order, (ROOF POLE WELL), or the order, (WELL POLE ROOF). There is no basis for selection. Hence, PATN follows a principle of least commitment, predicting the disjunction of the two invocation orders.

This application of the principle of least commitment is accomplished using a *chart*[4] of alternative plan derivations called the *planchart*. The planchart is similar to an *AND/OR goal tree* but involves a variety of node types and shares substructures economically. Figure II:2 illustrates how the two equally likely sequences are represented in the planchart. As PATN generates predictions, the required bookkeeping is performed by expanding this planchart.

Since the main steps for the two sequences are identical, they provide no evidence regarding ordering. The interfaces provide the critical evidence, so PATN solves the interfaces for one order, (WELL POLE ROOF). Because the choice is arbitrary, another demon is created to expand the (ROOF POLE WELL) order in case the interfaces fail to match. Except that TRIANGLE is already in the answer library, PATN has predicted the protocol of figure I:8.

Besides predicting PATN's default solution, three arbitrary choices have been flagged as likely failure points. If the specific discrepancy pattern monitored by one of the three corresponding demons is detected, that choice will be reconsidered. If non-specific mismatches are encountered, backup to other decisions will occur in the usual way. Note that most choices are not arbitrary and have *not* been flagged. (This helps to avoid the usual inefficiency

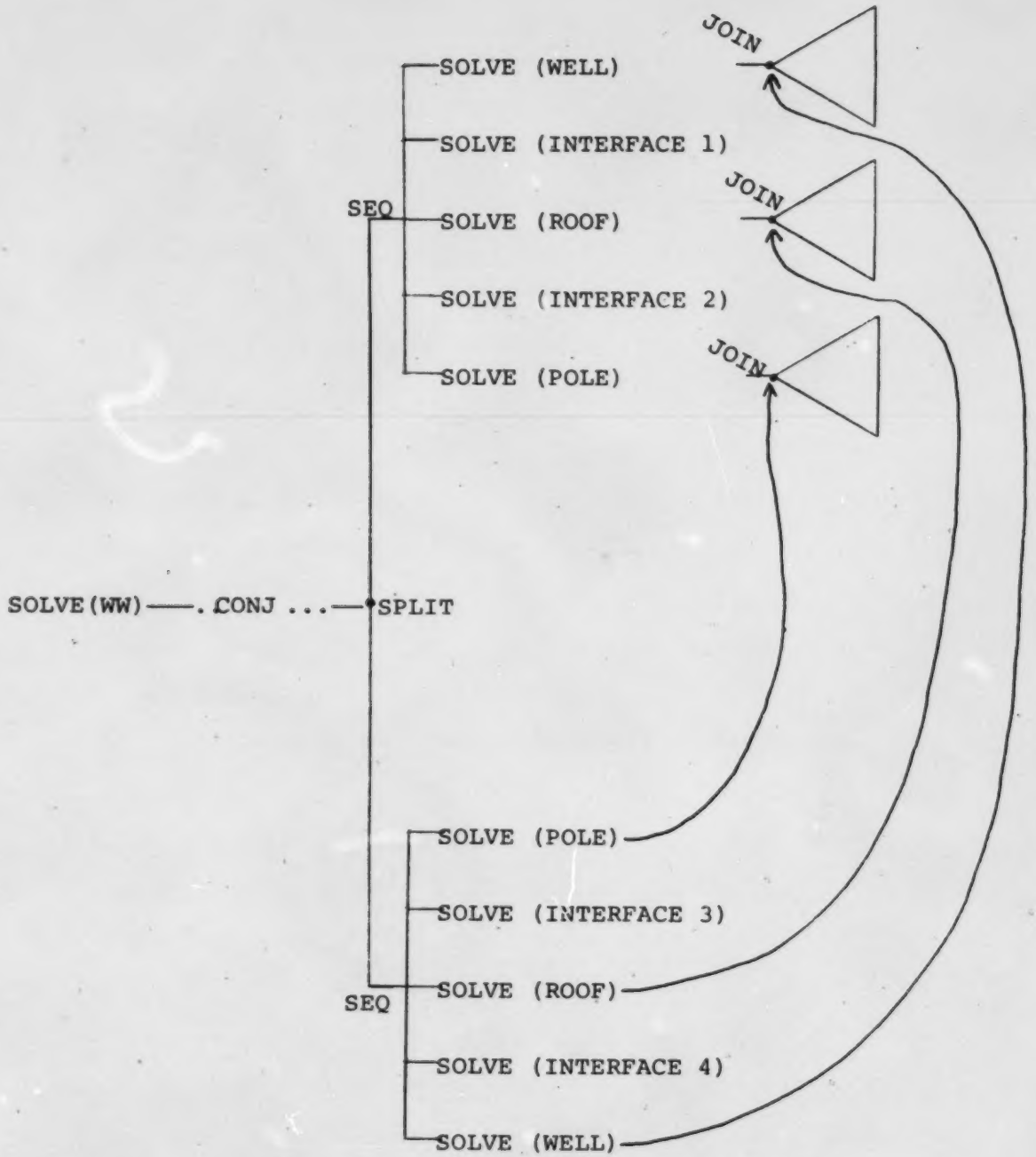


FIGURE II:2 SIMPLE PLANCHART FOR ALTERNATIVE ORDERS

associated with pure backtracking control structure: "failing in all possible ways.")

At this point, control passes to the bottom-up analytic routines.

6.2. Modifications Based on Bottom-up Evidence

PAZATN now attempts to interpret the first protocol event in a manner consistent with PATN's default solution.

E01 ?TO WELL

E01 is classified as a TO event -- Logo punctuation beginning a procedure definition. The event specialist for TO events is called upon to assign the event to some expectation.

PAZATN does not use mnemonic clues, and no significance is attached to the student's particular choice of the name WELL.[5] The TO specialist examines the planchart (figure II:2) for candidate subprocedures. There are expectations for the top level (WW), WELL, POLE and the two interfaces. The default solution order is top-down, so E01 is assumed to start WW. However, solution order is so variable that other interpretations are plausible. Consequently the interpretation *splits* into separate analyses for each.

Whereas the planchart is used to keep track of alternative expectations, a second chart, the *datachart*, is used to keep track of alternative associations between protocol events and expectations. PATN expands the planchart; PAZATN's *event interpreter* expands the datachart. At any given time, some of the *partial interpretations* in the datachart are considered to be *active*; the rest are *hung*.

For expository purposes, we will assume that only one partial interpretation is active at a time. Rather than pursuing several alternatives in parallel, we will merely record them in case the need to back up arises. Hence after the split is performed, E01 is assigned to be the TO for the top level

procedure, WW. The parent node of this interpretation, a generator for alternative interpretations of E01, is hung.

With E01 assumed to start WW, E02 is now processed.

E02 >10 REPEAT 4 [20 30]

This does not match the expectation for a definition of the top level WW procedure. Therefore, backup occurs to the most recent split (at E01). The only alternative that can account for E02, that E01 is the start of WELL, is activated (figure II:3).

The protocol matches this new interpretation through E05.

E03 >20 FORWARD 100
E04 >30 RIGHT 90
E05 >END

Ambiguity arises at E06.

E06 >TO WW

Since ROOF can be identified with TRIANGLE and WELL has already been found, this must be WW, POLE or an interface. The POLE and interfaces are apt to be solved by in-line code; furthermore, top-down order is the default preference. Hence, although E06 causes a split, WW is clearly chosen as the active interpretation. Next, E07 is examined.

E07 >10 TREE

Rather than matching WW's expectations for a setup or a call to WELL, E07 matches the discrepancy pattern for two active demons. One demon represents the possibility that the (ROOF POLE WELL) order was used; this would require TREE to be the setup for ROOF. The other demon represents a potential reformulation involving TREE. This second demon is highly specific for this evidence and is therefore triggered. Control returns to PATN with a request for a reformulated model in which TREE is a subgoal.

PATN regroups ROOF and POLE into TREE, and then expands for a solution to

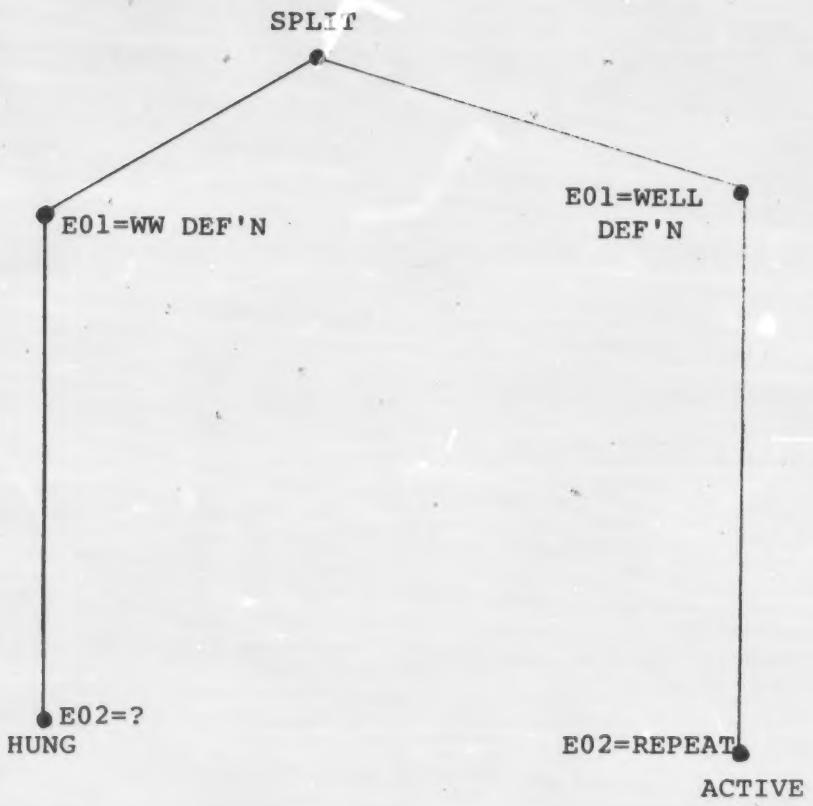


FIGURE II:3 DATACHART AT E02 OF WW

the revised model. When the sequential refinement loop is reached, the (TREE WELL) invocation order is chosen immediately on the basis of known protocol data. This need not have been PATN's choice from a problem solving point of view: this decision is forced by the bottom-up evidence. Figure II:4 shows the modified planchart.

After PATN has processed the reformulation request, E07 can be accommodated, as shown in the datachart of figure II:5. E08 is now examined. An interface is expected.

E08 >20 WELL

WELL is known to be a previously solved mainstep, violating that expectation. This is the standard pattern for an incomplete plan: an interface is expected but instead the next mainstep is found. A demon for incomplete plans is always active and is triggered by this situation. It passes control to DAPR which generates debugging expectations.

Each remaining event matches a DAPR expectation.

E09 >END
E10 >?WW
E11 >?EDIT WW
E12 >13 RIGHT 90
E13 >15 FORWARD 100
E14 >17 RIGHT 180
E15 >END

Hence the parse succeeds. Figure II:6 shows the final planchart and datachart, with marked nodes indicating the parse tree which is returned.

6.3. Some Informal Observations

We have hand-simulated PAZATN on about a half-dozen hypothetical protocols. This informal exercising of the design has led us to a number of tentative observations regarding PAZATN's capabilities. One question which arises is PAZATN's flexibility to handle alternative solutions. We are confident

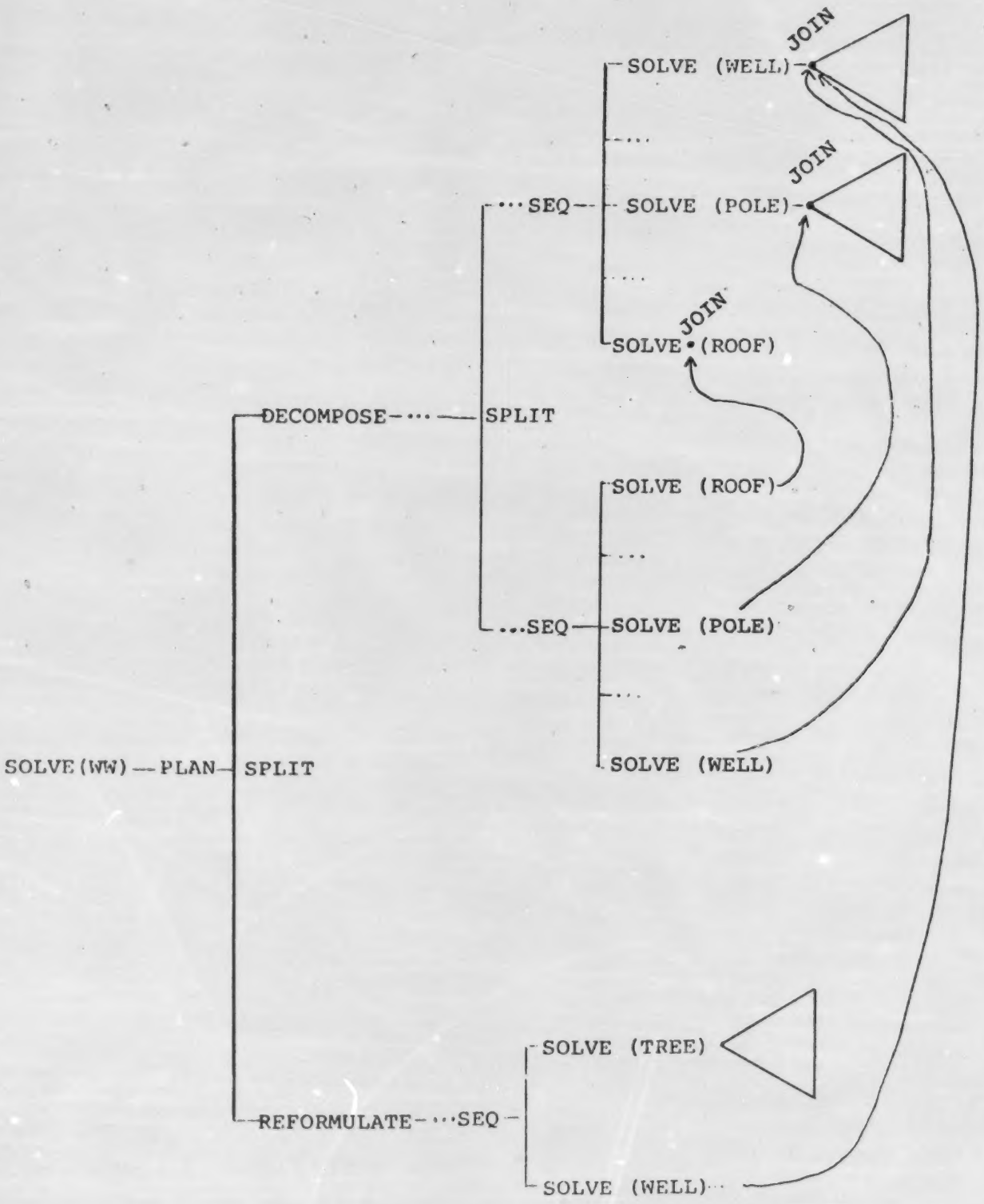


FIGURE II:4 PLANCHART AT E07, AFTER REFORMULATION

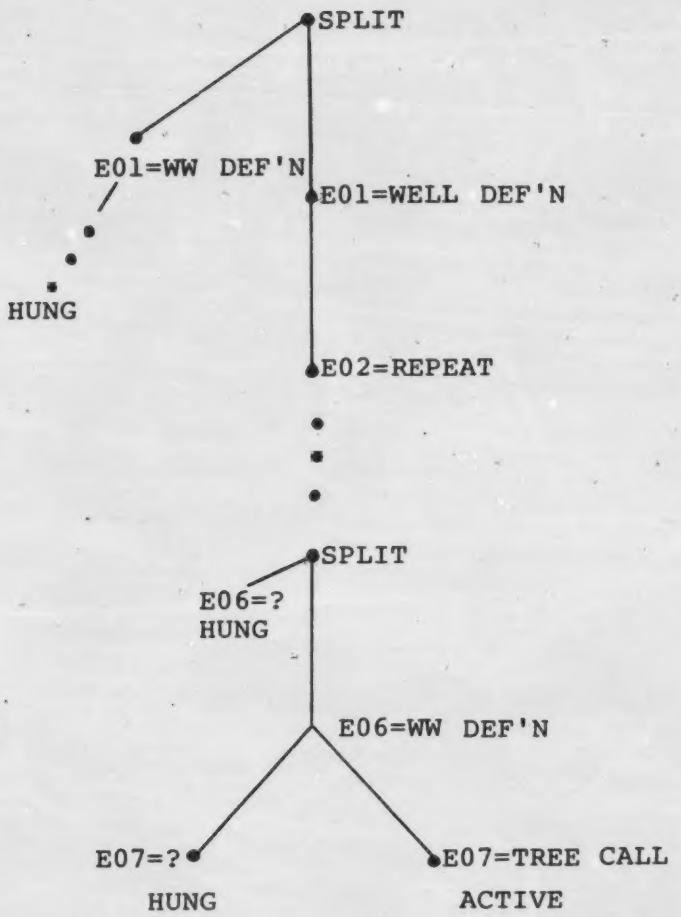
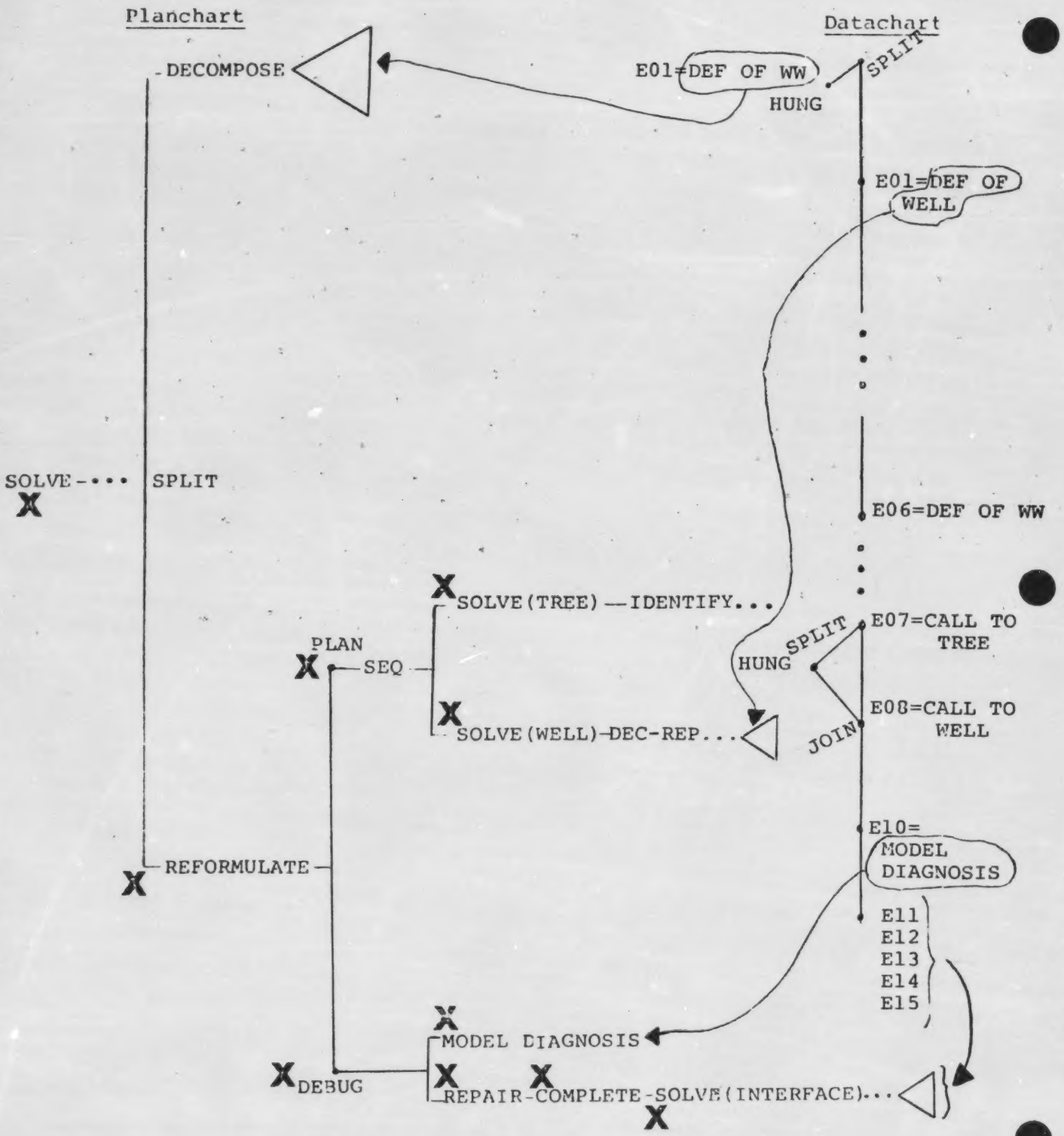


FIGURE II:5 DATACHART AT E07, AFTER REFORMULATION



Nodes marked by "X" are part of final parse.

FIGURE II:6 FINAL PLANCHART & DATACHART

that the following types of variation can be handled;

1. subprocedures versus in-line code;
2. incomplete plans where interfaces are solved by debugging;
3. incorrect plans where interactions are overlooked by the student (but known by PATN);
4. permutations of invocation or solution order;
5. standard reformulations (regrouping, generic-explicit conversion);
6. unnecessary nonlinear decompositions (accidental or for efficiency);
7. non-standard default parameters (FORWARD 75 as the basic unit);
8. simple forms of equivalence (BACK 100 versus FORWARD -100);
9. common errors such as mistyping, or omission of a line number.

On the other hand, the following types of variation pose problems for PAZATN:

1. interleaving of lines from different procedures if errors also occur in that a procedure is accidentally edited;
2. unrecognizable reformulations due to gaps in PATN's knowledge;
3. deliberately obscure code, or code involving many needless operations;
4. equivalence transformations resting on subtle domain theorems;
5. fully general recursion including heterarchical procedure calls.

Another observation concerns PAZATN's efficiency. For the simple protocols we have considered, after only a few false starts, PAZATN latches onto a correct set of expectations regarding the student's overall plan. After that point (which we would place at E08 for this protocol) interpretation of the remaining events proceeds without incident.

7. Organization of the PAZATN Protocol Parser

7.1. The Planchart
7.2. Representing Interpretations
7.3. The Datachart
7.4. Incremental Planchart Expansion
7.5. Markers and Marker Propagation
7.6. Preprocessing
7.7. The Event Classifier
7.8. The Event Interpreter
7.9. The Event Specialists
7.10. The Scheduler

In generating potential protocol interpretations, PATN is guided not only by *synthetic* evidence derived from examining the model, but also by *analytic* evidence derived from previously examined protocol events. If previous events have established that the student is pursuing a particular subgoal, then PATN will propose candidate solutions for that subgoal, even if it is not one which arises in PATN's preferred plan. Likewise if previous events have established that the student is pursuing a particular invocation order, then PATN will use that order in creating interfaces, even if another sequence leads to simpler interfaces. This sensitivity to the student's plan is accomplished by adding additional predicates to PATN's arcs which access assertions in the current partial interpretation.

This chapter presents the major PAZATN modules needed to use PATN in this analytic role. Chapter eight refines the discussion presented here.

7.1. The Planchart

PATN is an *intensional* representation of the plan space; there are a number of reasons for needing an *extensional* representation of the ATN process. Consequently a complete trace of PATN's operation, the *planchart*, is maintained. One reason for creating this data structure is to avoid repetitive calculations, but additional uses for the planchart will appear in the course of the

discussion. (Figure II:4 shows an example planchart from the analysis of WW.)

The planchart includes not only plans, but nodes of other types such as debugging episodes. As its name suggests, the planchart is a *chart* [Kay 1973; Kaplan 1973], a network which compactly represents alternative combinations of subexpressions. This economically represents PATN's partial solutions and their hierarchical annotation. Rather than generating the entire solution space at once -- which would be impractical even if it happened to be finite -- PATN expands this planchart incrementally as additional possibilities are needed by the analyzer.

Looking upward from a given *leaf*, the planchart resembles an AND/OR goal tree. However, there are a greater variety of node types, rather than just AND and OR. This allows the planchart to represent such concepts as whether conjunctive subgoals need to be accomplished in a specified order, or whether any order will do, allowing a greater variety of potential interpretations to be expressed parsimoniously.

The analysis process is closely tied to modifications of this data structure. In particular, the structural description assigned to a protocol corresponds to a pathway through the planchart starting from the root -- the top level SOLVE node -- to the individual protocol events corresponding to a subset of the leaves. The semantic variables and pragmatic assertions are generated by PATN along with the parse, and are attached to the corresponding planchart nodes.[6] Consequently, the structure building actions of the protocol parser are performed entirely by PATN.

7.2. Representing Interpretations

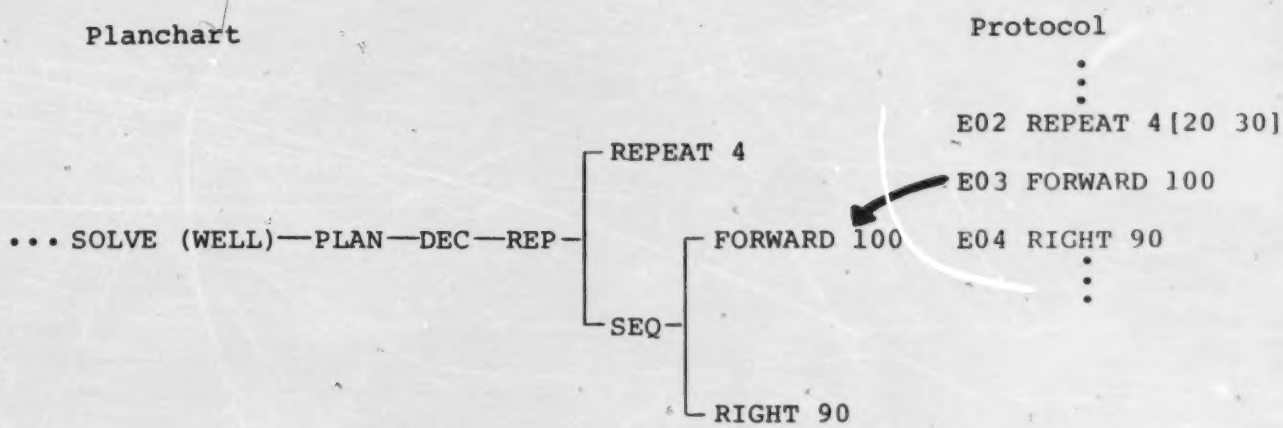
An *interpretation of an event* is represented as an assignment of that event to a leaf of the planchart (figure II:7). Similarly, an *interpretation of the protocol* is a complete association list of such event assignments. A *partial interpretation* is an association list containing assignments for a subset of the events in the complete protocol.

Because of the chart representation of plans, individual events can be assigned to a single leaf but remain ambiguous as to which plan they belong to. The assignment captures exactly what can be concluded from the event: no more and no less. All possible interpretations consistent with the data are carried along.

In order to be assigned to a given leaf of the planchart, it is *not* necessary for the protocol event to match identically. Data events are converted to canonical form before assignment, so that equivalent forms (e.g., LEFT 90 and RIGHT 270) are not distinguished. Non-equivalent assignments are also possible, representing the analyzer's judgment that the protocol event was intended to match the planchart leaf but contains either errors, such as mispellings or mistypings, or different default parameters where a range of values is acceptable.

7.3. The Datachart

A partial interpretation *splits* when it proposes more than a single planchart assignment for an event. Some method for keeping track of the analyzer's alternative partial interpretations is needed. It should take advantage of the fact that, following a split, the event interpretations prior to that split remain the same: the common ancestry should be preserved. Ideally interpretations which agree on events both *before* and *after* a split should share



E03 has been assigned to the planchart generic side for WELL.

FIGURE II:7 INTERPRETING AN EVENT

the same representation for them; this is called a *join*.

The datachart serves these functions. Like the planchart, the datachart is a chart, so that it can economically store common substructure. Suppose that two interpretations have identical assignments for the first M events, and then split. The split corresponds to a single node having two descendants. Assertions corresponding to the shared part of the interpretation are automatically inherited from the parent node (figure II:8).

Whenever a low plausibility event assignment occurs the following actions are performed:

1. An assertion is added at the current node, indicating which event assignment is about to be made. This ensures that the same possibilities will not be repeatedly pursued.

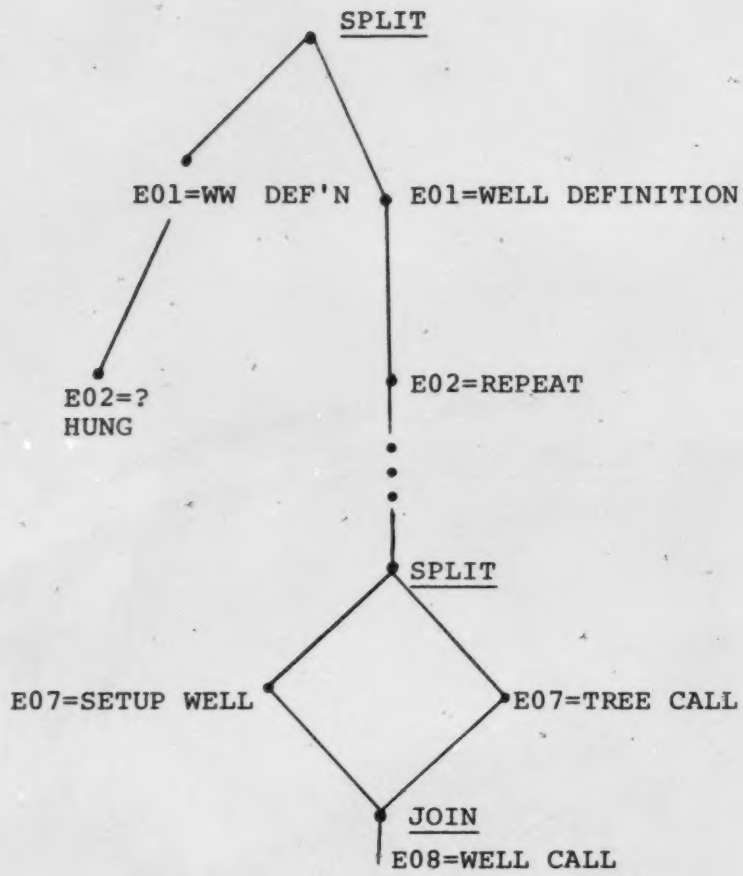
2. A new node is sprouted, which will inherit prior assignments from the parent node. This ensures that changes which reflect the uncertain assignment will not affect the state information of the parent node.

3. The uncertain assignment is performed at the new node. The normal operations associated with event interpretation (described below) are carried out.

4. The new node is placed on a list of NEW partial interpretations. This ensures that it will be scheduled for at least one cycle of further investigation.

5. The parent node is re-examined to determine if additional nodes should be sprouted representing alternative event assignments. If so, the above sequence of operations is carried out for each. When no further alternatives seem worth considering at the present time, the parent node is placed on a list of HUNG interpretations.

This technique has the feature that it is not necessary to explicitly list all of the possible alternative interpretations for a given event. After sprouting, the parent node no longer represents a *single* partial interpretation, but an *indefinite number* of implicit alternatives to its current offspring. Even after it is HUNG, the parent node contains the necessary state information to



The hypothesis that E01 starts the definition of WELL can be "seen" from the node for E08. Two possible explanations of E07 can also be seen.

FIGURE II:8 INHERITANCE OF DATA CHART ASSERTIONS

generate additional possibilities if these are ever needed.

7.4. Incremental Planchart Expansion

Consider the situation in which an active partial interpretation cannot find an acceptable planchart assignment for its next event. Two conclusions are possible: either *(a)* the current partial interpretation is a dead end, and should be moved to the HUNG list; or *(b)* the current partial interpretation is viable, but the planchart has not been expanded sufficiently to account for the current data.

This decision is crucial. If PAZATN is too miserly in allowing planchart growth, an event could be mis-interpreted as a deviant version of an existing leaf, when only slight growth would have allowed it to match a new leaf exactly. But if PAZATN is too eager to expand the planchart, the number of irrelevant solutions proposed could be enormous.

This decision is also very difficult, being complicated by the circumstance that data events need not identically match planchart leaves: they can differ because of postulated bugs or variant but acceptable parameter values (such as scale factors).

Four techniques are germane to this decision and its complications.

1. Protocol events are converted to a canonical form. This allows for handling simple forms of equivalence such as FORWARD -100 versus BACK 100.

2. Standard spelling correction procedures[7] are applied to unrecognized protocol events, using the fringe of the planchart as a dictionary. This allows for handling simple mistypings and mispellings.

3. A hash coding scheme uses the critical terms of an event (e.g., the *FORWARD*, but not the *100*) as keys.[8] This allows acceptable variants of events (e.g.,, those differing only by a scale factor) to be located.

4. The neighbors of a planchart leaf provide *expectations* which

influence the plausibility of event assignments to that leaf. The next section describes a scheme for generating these expectations.

7.5. Markers and Marker Propagation

A marker propagation technique helps in deciding whether to expand the planchart by providing a precise representation for expectations. Markers also determine the final protocol parse by selecting a pathway through the planchart. Assigning a protocol event to a planchart leaf *marks* that leaf. Three types of markers are used: *(1)* a standard marker for events that match identically or differ only in a flexible parameter value; *(2)* a distinguished marker for top down DEFINED plans prior to encountering the body of the subprocedure; and *(3)* a distinguished marker for deviant events involving mistypings or similar errors.

A constituent is *expected* to the extent to which finding it results in propagations, where propagation through the planchart is characterized by rules such as:

MPR-DISJ. If the parent of a marked node is disjunctive (i.e., a split), the parent is marked;

MPR-CONJ. If the parent of a marked node is conjunctive (e.g., SEQ) and every sibling of the marked node is marked, the parent is marked.

The rules shown here are incomplete. Top down DEFINED plans, for example, receive special treatment to ensure that after completing a superprocedure the expectations for its subprocedures remain in effect.

As an example of the use of these rules, consider a bottom-up DEFINED plan, where a subprocedure is first defined and then called by a superprocedure. After the subprocedure definition has been encountered, its use by some superprocedure is expected. The planchart would contain a marked SOLVE node for the subprocedure and an unmarked USE node for its use in the other procedure,

both dominated by an unordered conjunctive "&" node (figure II:9). The USE is expected because marking its node would result in a propagation at least as far as the SOLVE dominating the DEFINED node.

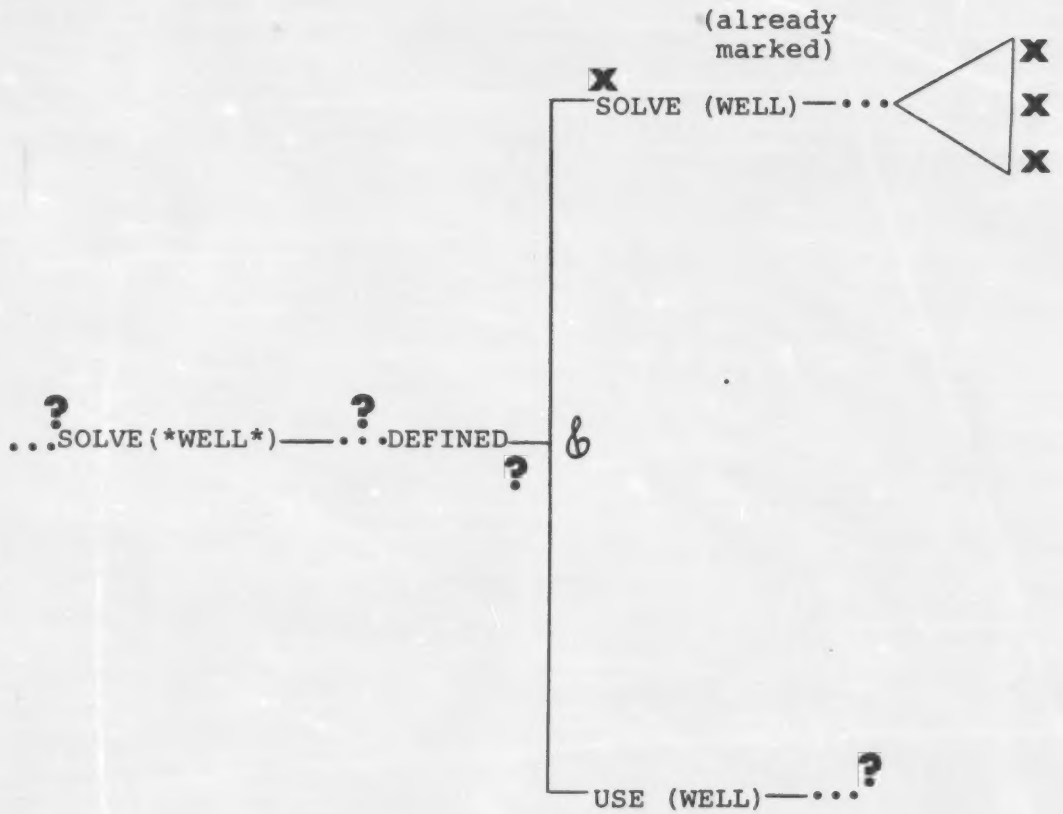
Suppose that an expectation (such as the bottom-up DEFINED plan example) fails to be satisfied after many events. One possibility is that the partial interpretation which expects it is completely wrong, and should be abandoned. A second possibility is that the partial interpretation is basically correct, but the student has accomplished the expected effect in an alternative way (e.g., incorporated the subprocedure's definition in-line instead of calling it as expected). This second case turns off the expectation, since it becomes dominated by a marked node (figure II:10).

A third possibility is that the student accidentally left out the relevant line of code. This is detected when protocol events indicate that the episode is finished. In the Logo world this corresponds to encountering the END statement for the superprocedure. END statements force propagations even when some expectations are not satisfied; but the plan is flagged as incomplete, debugging expectations are generated, and the plausibility is lowered. If the debugging predictions are then confirmed, the plausibility is restored and the expectation considered satisfied.

Markers, as a representation for expectations, provide evidence regarding the plausibility of interpretations, which is especially useful when planchart expansion is under consideration. Typical plausibility guidelines include:

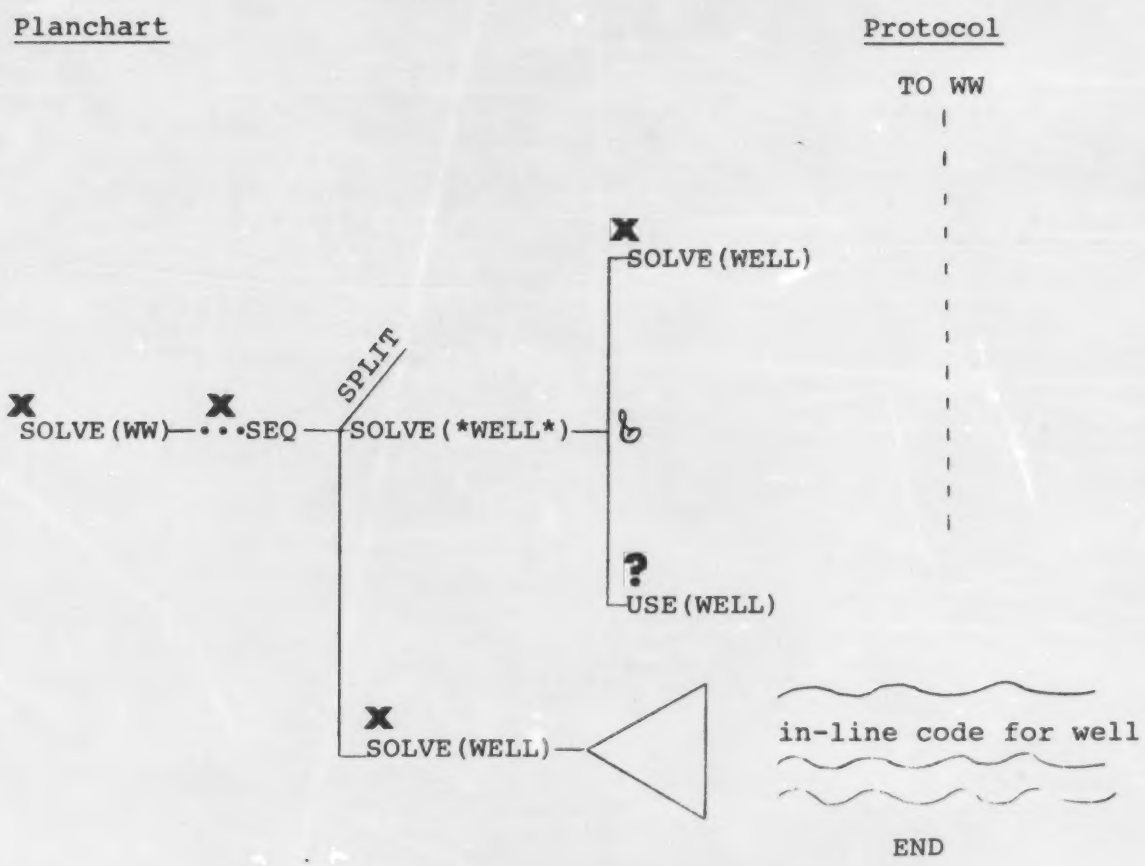
PLG-1. Event assignments that result in longer chains of propagations are more plausible than those that result in shorter chains of propagations or none at all.

PLG-2. Interpretations that leave few expectations unsatisfied are more plausible than those that leave many expectations unsatisfied.



A use of WELL is expected because it would cause the propagations shown as ?'s.

FIGURE II:9 DEFINED PLANS: AN EXAMPLE OF PROPAGATION AS EXPECTATION



This use of WELL is no longer expected, since it is now dominated by a marked node.

FIGURE II:10 EXPECTATIONS CANCELLED, DOMINATED BY MARKED NODE

7.6. Preprocessing

PAZATN includes a preprocessor which performs four functions.

1. Low level syntactic anomalies such as typographical errors corrected using the RUBOUT and BREAK keys are filtered out; only the corrected versions of such events are examined.

2. Low level segmentation clues are noted. For example, with raster scan TV TURTLES [Lieberman 1976] global connectivity of vectors is readily detectable and suggests a segment boundary.

3. Timing data are collected. This information may be of value in testing psychological claims, and in some instances the plausibility of an interpretation depends upon the elapsed time between type-ins.[9]

4. The primary syntactic class of each event is recorded to avoid recomputing it under each interpretation. Classification is performed by a separate module which can be re-invoked if the primary class is later called into question.

7.7. The Event Classifier

The event classifier, one of the few PAZATN modules which must be redefined for each domain, contains the syntactic knowledge necessary to distinguish various domain-specific event types. For the programming world, the event types include RUN events, EDIT events, and so on. In assigning an interpretation to an event, a variety of semantic and pragmatic evidence is ultimately considered by PAZATN, but the event classifier is restricted to syntactic evidence.

The event classifier can be invoked in three modes. Normally it is invoked by the preprocessor, with its input an event and its output the event's *primary syntactic class*; for most events, this is sufficient. In the second mode it is invoked by partial interpretations which question the primary syntactic class, with a specific alternative class being considered. Here its input is an event and a class name; its output is a numerical score summarizing the syntactic evidence supporting the alternative class. In the third mode the

classifier is invoked by partial interpretations which question the primary syntactic class but with no specific alternative class under consideration. An exhaustive rank ordered list of categories and scores is returned.

Event classification will be performed using straightforward pattern matching. No further details are given here.

7.8. The Event Interpreter

The event interpreter is responsible for category independent operations of event interpretation. This includes the node sprouting sequence described in the datachart section, the processing required for marker propagation, and the plausibility computations. The rationale for grouping these activities is modularity: they are required for every category of event interpretation.

The event interpreter is PAZATN's inner loop. It is invoked by the scheduler with two arguments: a partial interpretation, and a protocol event. It attempts, in cooperation with one or more event specialists, to account for the protocol event in the context of the partial interpretation. This can result in the creation of additional (descendant) partial interpretations. Control returns to the scheduler when event interpretation is complete.

7.9. The Event Specialists

A collection of domain specific *event specialists* (ESP's) are responsible for category dependent operations of event interpretation. Each specialist contains the requisite knowledge for analyzing events of a particular syntactic type. The event interpreter invokes an ESP, in the context of a partial interpretation, with an event and an implicit assumption regarding its syntactic category. The specialist is free to assign any interpretation to the event which is consistent with the category.

If the event specialist does not return with a sufficiently plausible

event assignment, the event interpreter then considers the possibility that the syntactic category postulated for the event is incorrect. Whenever an event is interpreted as being in error, expectations for diagnosis and repair are generated by DAPR at the request of the event interpreter.

ESP's use a decision tree organization to factor the analysis into several cases. Each case represents an assumption about intent; if the assumption is uncertain, the state of the interpretation is preserved by sprouting a new datachart node. This is exemplified by the Logo ADDCODE ESP, whose flowchart appears in figure II:11.[10]

The ADDCODE classification assumes that the current event is intended to add a new line of code to the procedure definition. Hence it must be determined whether Logo is actually in definition mode. If not, the following event will be an error message. If the ADDCODE assumption is correct despite the error, the current event will be repeated after a TO event.

Lookahead is required to assign the current event to be an erroneous version of a later event. However, in a real time tutoring application, the later event might not have occurred yet; moreover, processing more than one event would exceed the scheduler's resource allocation. This dilemma is resolved by creating a demon to represent the current event assignment. The demon will fire when the future event is assigned, assigning the now current event to be a deviant version of the later event.

In the case where Logo is in definition mode, ADDCODE branches to one of the following subcases: *(a)* the added code is a turtle primitive; *(b)* the added code is a Logo control statement (such as a recursion or iteration line); *(c)* the added code is a call to a user procedure other than a recursion line.

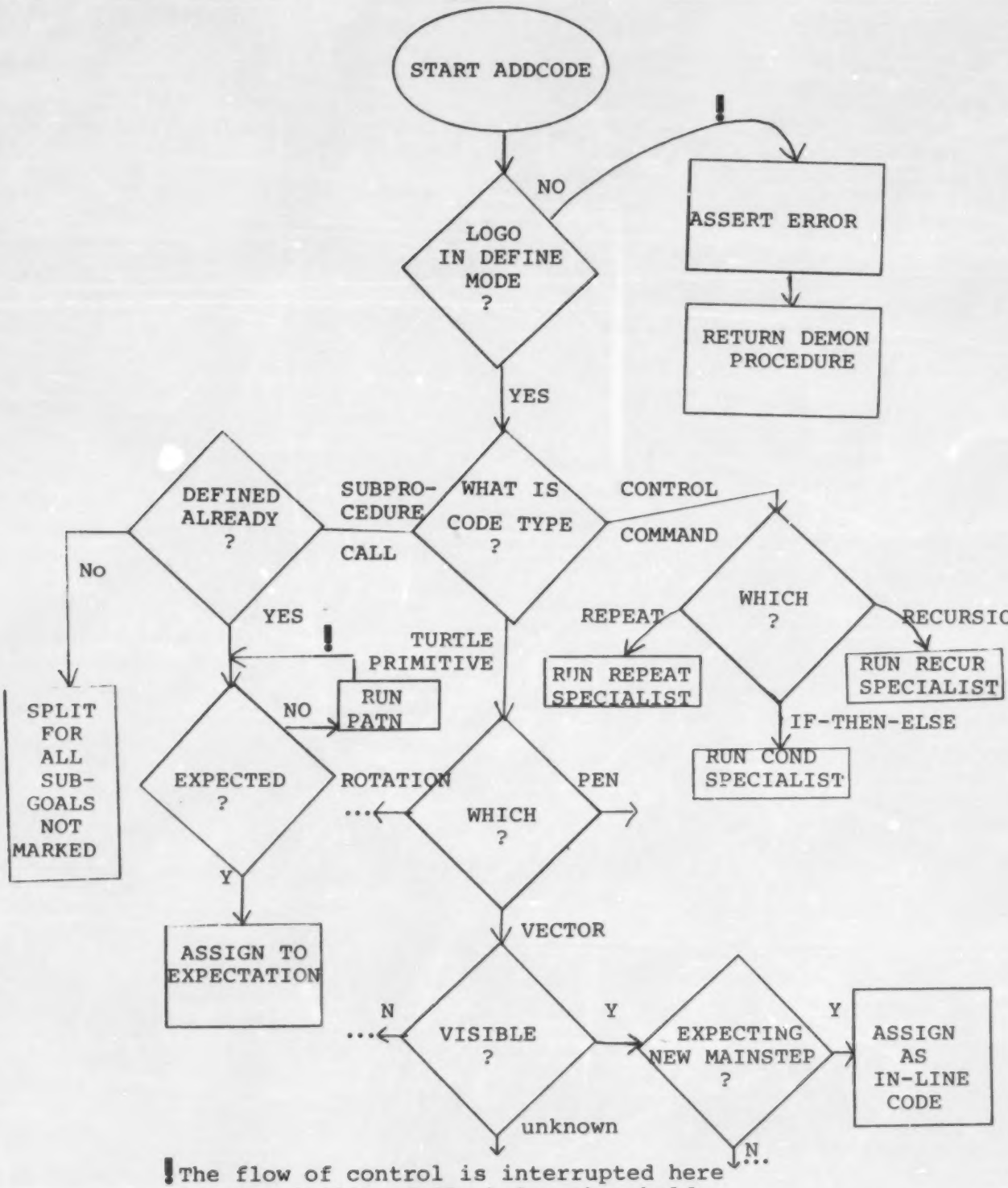


FIGURE II:11 FLOWCHART FOR ADDCODE ESP

7.10. The Scheduler

The scheduler's task is to cause partial interpretations which have a reasonable likelihood of succeeding to make progress, and prevent those that are likely to fail from consuming valuable resources. Operationally this means driving PAZATN through a *best first coroutine search* of the space of partial interpretations.

The search is accomplished by maintaining three lists of partial interpretations: NEW, ACTIVE, and HUNG. The scheduler cycles through the ACTIVE list, allowing each item to process one protocol event. Then the plausibility of each modified interpretation is recomputed, and the ACTIVE and HUNG lists are re-chosen. NEW interpretations, which result from the splitting of ACTIVE interpretations on the previous cycle, are then moved to the ACTIVE list, guarantying them at least one quantum of processing. The plausibility of a partial interpretation increases with each additional event accounted for. (This acts to decrease the relative plausibility of older HUNG interpretations.)

This process continues until at least one ACTIVE interpretation has processed the last input event without unsatisfied expectations. If the first successful interpretation is not sufficiently better than every other candidate, some of the better alternatives are pursued until they become implausible or determine that the protocol may successfully be interpreted in more than one way.

8. Refining the Protocol Parser

8.1. Lookahead
8.2. Least Commitment
8.3. Differential Diagnosis

Our basic protocol parsing scheme is to generate expectations with PATN and then try to match these expectations to a protocol. This process is refined by several techniques which have enhanced the effectiveness of problem solving and language processing programs: *lookahead* (e.g., [Aho & Ullman 1972]), *least commitment* (e.g., [Sacerdoti 1975]) and *differential diagnosis* (e.g., [Rubin 1975]).

8.1. Lookahead

Lookahead and *least commitment* are related search strategies designed to avoid premature decisions based on inadequate evidence, which can result in needless backup. Lookahead consists of briefly examining subsequent input events before interpreting the current event.

PAZATN can accomplish a limited form of lookahead by using demon procedures to represent event assignments. When the current event assignment depends upon a future event assignment, a demon is created which will complete the current assignment when the missing evidence from the future assignment is available.

8.2. Least Commitment

Variability in solution order exemplifies the need for avoiding premature commitments. PATN always defines the top level plan before expanding subproblems, representing strict top-down problem solving (figure II:12), but human programming is rarely this uniform. When the need for a particular subgoal has been established, it may be expanded immediately, prior to completing the top

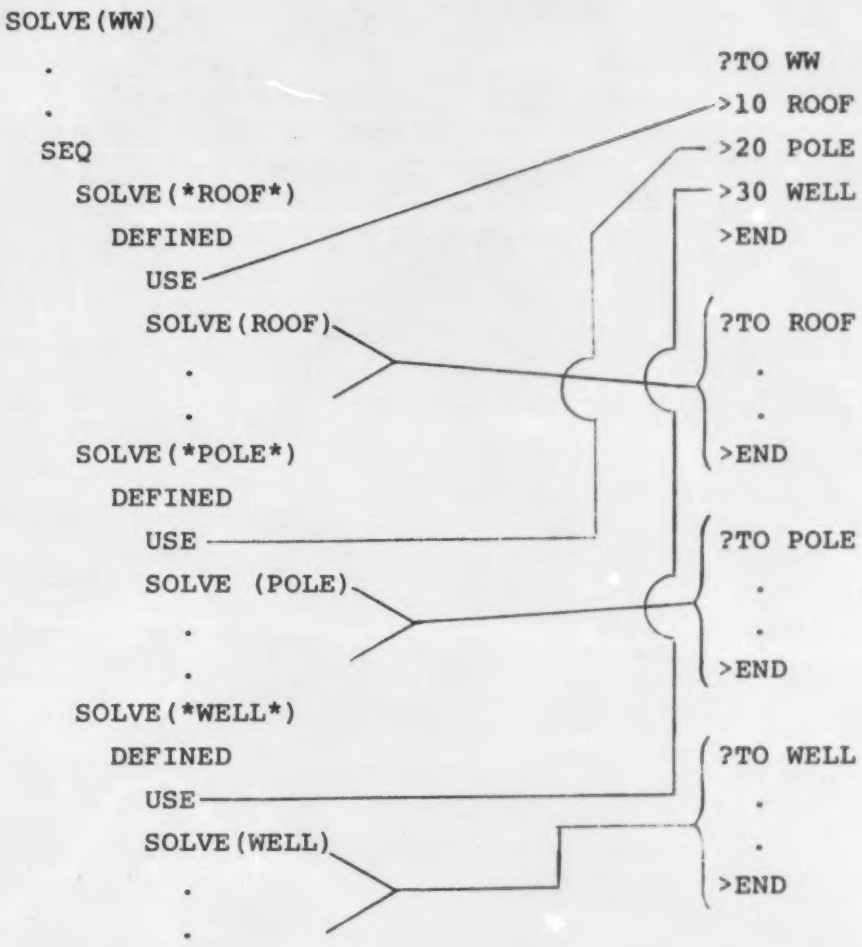


FIGURE II:12 A TOP-DOWN EXPANSION FOR WISHINGWELL

level plan, representing bottom-up problem solving (figure II:13).

Least commitment helps to minimize misleading mismatches between planchart and protocol resulting from different solution orders. This is accomplished by using unordered conjunctive "&" nodes in the planchart. Thus when DEFINED plans are expanded to USE & SOLVE, the SOLVE may occur prior to the USE with no loss in plausibility.

The least commitment policy is applied to variability in invocation order as well. When, as was the case with WW, more than one invocation order is acceptable, the planchart is split. This parallels the use of procedural nets [Sacerdoti 1975] to avoid overspecifying ordering constraints (figure II:14). The chart data structure allows the ambiguity to be represented without significant additional cost: if the mainsteps are identical for both orders, then two copies will *not* be stored.

Despite its virtues, though, least commitment could be overdone, resulting in so large a disjunction of expectations that no guidance would be obtained. PAZATN strikes a balance between overcommitting itself and refusing to take decisive action: it avoids arbitrary choices in the course of a given decomposition strategy, but adheres to a given formulation of the model unless required to change it by specific bottom-up evidence.

8.3. Differential Diagnosis

The use of demon procedures to implement lookahead was discussed earlier. Another use of demons is to perform *differential diagnosis*, using highly specific clues to distinguish between similar competing interpretations. The primary application of differential diagnosis demons is to the choice between assigning an event to one of an existing disjunction of expectations, and reformulating the problem description in response to bottom-up evidence.

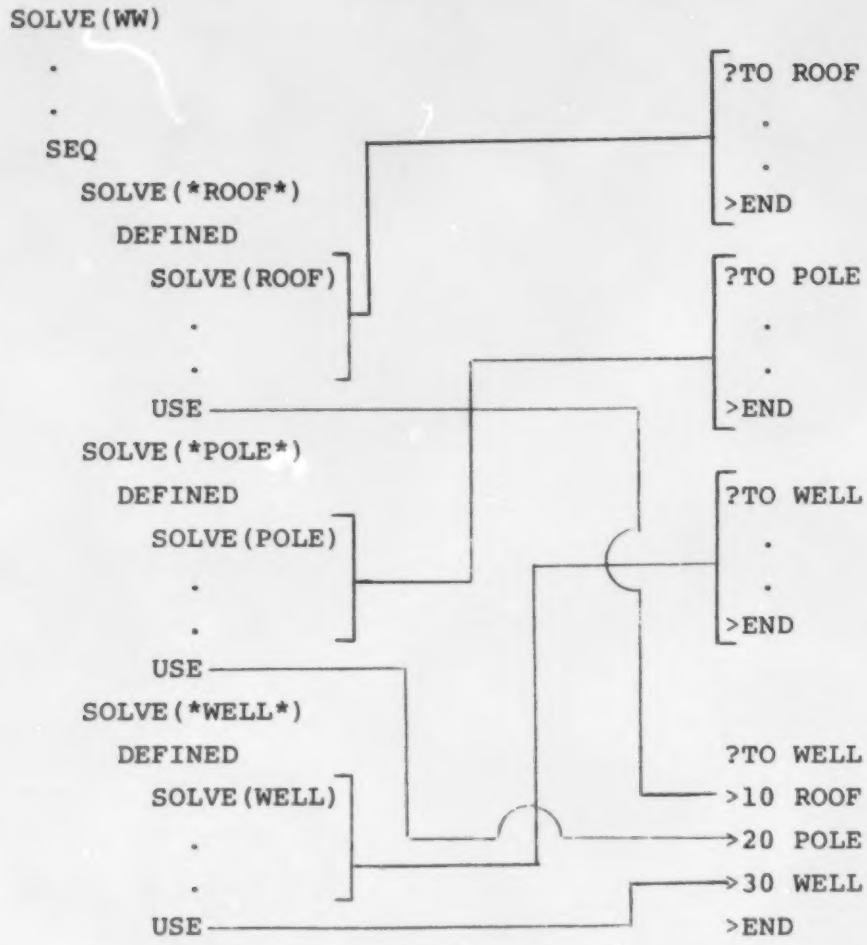


FIGURE II:13 A BOTTOM-UP EXPANSION FOR WISHINGWELL

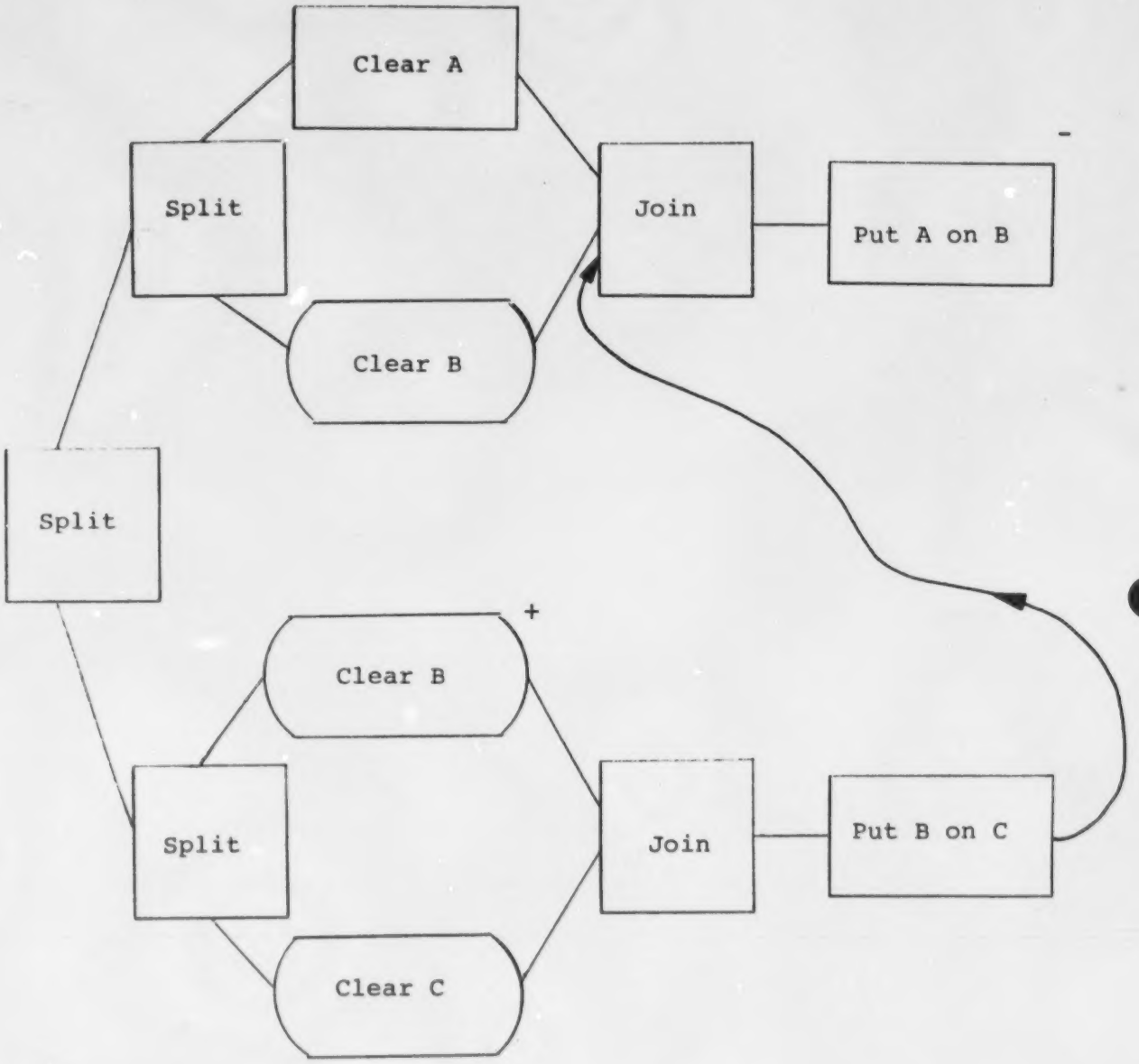


FIGURE II:14 A PROCEDURAL NET FOR BUILDING A TOWER AFTER CRITICISM TO RESOLVE CONFLICTS

[BASED ON SACERDOTI, 1975, p. 15]

To illustrate this, we present one example of a complementary pair of *demon templates*. These templates can be instantiated to realize differential diagnosis behavior in specific situations.

A. If the current code segment (or its picture) matches a disjunctive subset of the current expectations, select that subset.

B. If no expectation matches the current code segment (or its picture), consider a reformulation using the segment's effect as a subgoal.

9. Conclusion

9.1. Recapitulation
9.2. Implementation Plans

9.1. Recapitulation

In this report we have investigated the problem of analyzing elementary problem solving protocols. The result of this investigation is the design for PAZATN, a domain independent protocol parsing scheme, which was applied to the Logo graphics programming domain. Coupled with the Logo ESP's, the design was sufficiently well-specified that PAZATN could be hand-simulated for a simple example with encouraging results. The foundation for the approach was SPADE, a linguistic theory of design in which problem solving is viewed as a structured process of planning and debugging. This led us to the definition of an *interpretation* as a parse tree augmented by semantic and pragmatic annotation associated with each node.

A key ingredient in the design is a machine problem solver called PATN. PATN employs an augmented transition network to represent fundamental planning concepts, including techniques of identification, decomposition, and reformulation. Considerable leverage is obtained from PATN's ability to generate successively less preferable solution paths, by a series of pragmatically guided planning decisions, as well as from PATN's characterization of certain bugs as errors in these planning choices.

We found an analogy to computational linguistics to be fruitful, providing insights into data representations and search strategies which are characteristic of research in syntactic analysis [Kay 1973; Kaplan 1973] and speech recognition (e.g., [Lesser et al. 1975; Paxton & Robinson 1975]). For example, the *chart* representation is used to economically store well-formed

substructures. *Lookahead*, *least commitment*, and *differential diagnosis* are example strategies used to refine PAZATN's search for a parse. These allow for proceeding on the basis of reasonable assumptions when necessary, while retaining the ability to modify the interpretation in response to anomalies.

The analysis procedure has been designed to obtain maximal advantage from both top-down guidance from the task description and bottom-up protocol evidence. Analysis proceeds by a *best first coroutine search* of a space of partial interpretations. The planchart, a data structure resembling an AND/OR goal tree, is used to keep track of expectations. By careful selection of the representational scheme, this structure achieves considerable storage economy. Partial knowledge of structure and of the status of expectations is recorded using a scheme of planchart markings and marker propagations. The planchart is incrementally expanded by PATN when existing expectations are inadequate in view of the protocol data. A second chart, the datachart, is used to keep track of the state of alternative partial interpretations.

Although PAZATN is not yet a working program, the design is sufficiently specific so as to be hand-simulable. In hand-simulation, there is a danger of unintentionally drawing upon knowledge which has not been isolated or formalized. Care was exercised to avoid this pitfall, and the examples are encouraging evidence that the approach is fundamentally sound. Still, hand-simulation is not seen as a substitute for implementation. The next phase of the research is to implement and experiment with a prototype analyzer.

PAZATN is a generalization and extension of previous approaches. PAZATN grew out of Goldstein's [1974; 1975] plan-finder for MYCROFT. The differences are that PAZATN: *(a)* generates interpretations consistent with the recently developed SPADE theory; *(b)* handles the wider range of event types necessary to analyze protocols rather than finished programs; *(c)* takes advantage of the

dynamic information in these additional event types regarding subgoal structure and development; and *(d)* is not limited to the Logo domain. The SPADE theory developed from the MYCROFT theory of program understanding as well as related work by Sussman [1973], Papert [1971] and Sacerdoti [1975]. [Goldstein & Miller 1976b] argues that SPADE represents progress over this earlier theorizing. PAZATN also complements the features and limitations of analyzers developed at Carnegie-Mellon University. The major theoretical advance is a highly structured model of program synthesis. The major practical advance is the modularization of domain specific knowledge, which indicates that the PAZATN framework ought to be applicable to a wide variety of task domains.

PAZATN is independent of the detailed form of the synthetic formalism: it does not intrinsically depend on PATN being an augmented transition network. It is only necessary that the synthetic component plan and debug by making a series of pragmatic choices which can be summarized by the planchart data structure, and that it be capable of generating not one, but an entire space of progressively less favored solution paths. Finally, an implicit assumption runs throughout the analyzer's design that solutions can be decomposed into syntactic, semantic, and pragmatic elements. It may be that any synthetic formalism satisfying these constraints is trivially equivalent to an ATN. Such questions are notoriously difficult to settle.

However, an important issue in the design is the breadth of the synthetic theory. There are of course particular omissions such as conditional plans, which have been deliberately -- but only temporarily -- ignored. The greater threat comes from the unknown. Even very young children display incredible richness in their problem solving. Although SPADE's origins are partly empirical, crucial phenomena -- perhaps those most in need of investigation -- may have been overlooked. This remains a topic for investigation.

9.2. Implementation Plans

There are several ways in which an apparently sound design could fail in implementation. The space of partial interpretations could turn out to be very large relative to the sources of constraint which have been isolated. The variety of knowledge used by human programmers could greatly surpass our current estimates. PAZATN's storage requirements could exceed practical bounds. The analyzer could be too rash in its heuristic quest for efficiency, terminating prematurely with unacceptable interpretations. Too great a demand could be placed on PATN's ability to find reformulations encompassing bottom up evidence. Hand-simulations or even partial implementations could overlook such impediments.

Consequently, complete implementation of a prototype system is essential for validating the research. We intend to perform this implementation incrementally, beginning with an interactive version. At first, only straightforward bookkeeping functions will be automated. The computer will record plans and event assignments using the two charts, but decisions regarding which interpretations to pursue will be made by a human investigator. This will be replaced by a version which performs the routine analysis of most events, only requesting help on more difficult cases. Eventually, the analysis will be handled completely by the machine. A modeling component for inducing personalized ATN's will be implemented, and its predictive power explored. To demonstrate PAZATN's understanding of the protocols, a question answering module using a formal query language will be constructed to operate over PAZATN's output.

10. Notes to Book II

1. [Brown et. al. 1977] includes a chapter indicating the direction of our efforts to apply PAZATN to the symbolic integration domain.

2. The one exception to this is that some irrational errors (such as mistypings) can be recognized as unsuccessful manifestations of PATN plans.

3. This use of demons is inspired by Charniak's [1972] work on story understanding. Demons are a type of *antecedent theorem* [Hewitt 1972].

4. The chart data structure is due to Kay [1973] and Kaplan [1973]. *Generalized AND/OR graphs* [Levi & Sirovich 1976] are data structures similar to our plancharts derived on the basis of independent formal considerations. We think that, because of the extension from trees to charts as well as the incorporation of a larger variety of node types, our plancharts are an improvement over generalized AND/OR graphs along the dimensions of generality, storage economy, and expressive power.

5. We intend to provide PAZATN with limited heuristics for recognizing mnemonic identifiers. However, relying on user chosen names for guidance in general would be too unreliable. Hence, to emphasize that such guidance can be dispensed with, we assume here that procedure names are unrecognizable.

6. In assigning a protocol event to a planchart leaf, the type of event and the value of :MODEL are considered, but the other semantic variables and the pragmatic assertions are generally not considered. This is a simplification which ignores the possibility for complex semantic and pragmatic ambiguities. For example, two interpretations might be identical except for the value of :ADVICE at some node. Although this difficulty seems unlikely, PAZATN could be elaborated slightly to handle it. Here we ignore the problem and show only the structure description and the name of the submodel in our diagrams. (The other variables and the pragmatic assertions, being assumed unambiguous, are suppressed.)

7. Interlisp [Teitelman 1974, pp. 17.10-17.14] provides such a spelling corrector. See also [Teitelman 1970].

8. Such techniques are in common use. See, for example, [Greenblatt et al. 1967, pp. 806-807].

9. For example, if much more than the typical time elapses between the type-in of two consecutive events, it is more plausible to interpret the second event as initiating a new episode. A more specific example involves the frequent errors associated with Logo line numbers. There are two such errors: *(1)* failing to include a line number when it is needed; and *(2)* accidentally including a line number when it is not needed. Consider the second. If much more than the typical time elapses between the type-in of the line number and the type-in of the remainder of the event, it becomes more plausible to interpret the event as a buggy RUN event rather than a legal but inexplicable EDIT event. Rather than storing every value, however, the preprocessor will accumulate summary statistics, only recording the specific data for type-ins which are

markedly slower than the average.

10. Information concerning other Logo event specialists as well as additional parsed protocols will be supplied by the authors upon request.

References

[Aho & Ullman 1972]
Aho, Alfred V., and Jeffrey D. Ullman, *The Theory of Parsing, Translation, and Compiling*, Volume I: Parsing, Englewood Cliffs, New Jersey, Prentice-Hall, 1972.

[Bhaskar & Simon 1976]
Bhaskar, R., and Herbert A. Simon, "Problem Solving in Semantically Rich Domains: An Example from Engineering Thermodynamics" (draft of paper to appear in *Cognitive Science*), Carnegie-Mellon University, C.I.P. Working Paper 314, February 24, 1976.

[Brooks 1975]
Brooks, Ruven, *A Model of Human Cognitive Behavior in Writing Code for Computer Programs*, Carnegie-Mellon University, Report AFOSR-TR-1084, May 1975.

[Brown et al. 1974]
Brown, John Seely, Richard R. Burton and Alan G. Bell, *SOPHIE: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting (An Example of AI in CAI)* (Final Report), Bolt, Beranek and Newman, Report 2790 (Artificial Intelligence Report 12), March 1974.

[Brown et al. 1975]
Brown, John Seely, Richard R. Burton, Mark L. Miller, Johan DeKleer, Steven Purcell, Catherine Hausmann and Robert Bobrow, *Steps Toward a Theoretical Foundation for Complex Knowledge-Based CAI* (Final Report), Bolt, Beranek and Newman, August 1975.

[Brown et al. 1977]
Brown, John Seely, Richard R. Burton, Catherine Hausmann, Ira P. Goldstein and Mark L. Miller, *Intelligent Tutoring Systems: Experiments and Theory*, Bolt, Beranek and Newman, ICAI Report 4, 1977.

[Burton & Brown 1976]
Burton, Richard R., and John Seely Brown, "A Tutoring and Student Modelling Paradigm for Gaming Environments," in R. Colman and P. Lorton Jr. (eds.), *Computer Science and Education* (Advance Proceedings of the Association for Computing Machinery Special Interest Groups on Computer Science Education and Computer Uses in Education Joint Symposium, Anaheim, Cal.), SIGCSE Bulletin, Volume 8, Number 1 (SIGCUE Topics Volume 2), February 1976, pp. 236-246.

[Charniak 1972]
Charniak, Eugene, *Toward a Model of Children's Story Comprehension*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Technical Report 266, December 1972.

[Chomsky 1965]
Chomsky, Noam, *Aspects of the Theory of Syntax*, Cambridge, Massachusetts, The M.I.T. Press, 1965.

[Goldstein 1974]
Goldstein, Ira P., *Understanding Simple Picture Programs*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Technical Report 294, September 1974.

[Goldstein 1975]
Goldstein, Ira P., "Understanding Simple Picture Programs," *Artificial Intelligence*, Volume 6, Number 3, Fall 1975.

[Goldstein 1976]
Goldstein, Ira P., *A Preliminary Proposal for Research on The Computer as Coach: An Athletic Paradigm for Intellectual Education*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 389 (Logo Memo 37), December 1976.

[Goldstein & Carr 1977]
Goldstein, Ira P., and Brian P. Carr, *Overlays: A Theory of Modelling for Computer Aided Instruction*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, forthcoming Memo, February 1977.

[Goldstein & Miller 1976a]
Goldstein, Ira P., and Mark L. Miller, *AI Based Personal Learning Environments: Directions for Long Term Research*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 384 (Logo Memo 31), December 1976a.

[Goldstein & Miller 1976b]
Goldstein, Ira P., and Mark L. Miller, *Structured Planning and Debugging: A Linguistic Theory of Design*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 387 (Logo Memo 34), December 1976b.

[Greenblatt et al. 1967]
Greenblatt, Richard D., Donald E. Eastlake III, and Stephen D. Crocker, "The Greenblatt Chess Program," *Proceedings of Fall Joint Computer Conference*, Volume 31, Montvale, New Jersey, American Federation of Information Processing Societies, 1967.

[Heidorn 1975]
Heidorn, George E., "Augmented Phrase Structure Grammars," *Theoretical Issues in Natural Language Processing*, Cambridge, Mass., Association for Computational Linguistics, June 1975.

[Hewitt 1972]
Hewitt, Carl, *Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Technical Report 258, April 1972.

[Kaplan 1973]
Kaplan, Ronald M., "A General Syntactic Processor," in Randall Rustin (ed.), *Natural Language Processing*, Courant Computer Science Symposium 8 (December 20-21, 1971), New York, Algorithmics Press, 1973, pp. 193-241.

[Kay 1973]
Kay, Martin, "The MIND System," in Randall Rustin (ed.), *Natural Language Processing*, Courant Computer Science Symposium 8 (December 20-21, 1971), New York, Algorithmics Press, 1973, pp. 155-188.

[Lesser et al. 1975]
Lesser, V.R., R.D. Fennel, L.D. Erman and D.R. Reddy, "Organization of the Hearsay II Speech Understanding System," in *IEEE Transactions on Acoustics, Speech and Signal Processing*, Volume Assp-23, Number 1, February 1975, pp. 11-24.

[Levi & Sirovich 1976]
Levi, Giorgio, and Franco Sirovich, "Generalized AND/OR Graphs," *Artificial Intelligence*, Volume 7, Number 3, Fall 1976, pp. 243-259.

[Lieberman 1976]
Lieberman, Henry, "The TV Turtle: A Logo Graphics System for Raster Displays," *Proceedings of SIGGRAPH/SIGPLAN Symposium on Graphics Languages* (Joint issue of SIGPLAN Notices & Computer Graphics), April 1976.

[Miller & Goldstein 1976a]
Miller, Mark L., and Ira P. Goldstein, *Overview of a Linguistic Theory of Design*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 383 (Logo Memo 30), December 1976a.

[Miller & Goldstein 1976b]
Miller, Mark L., and Ira P. Goldstein, *Parsing Protocols Using Problem Solving Grammars*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 385 (Logo Memo 32), December 1976b.

[Miller & Goldstein 1976c]
Miller, Mark L., and Ira P. Goldstein, *SPADE: A Grammar Based Editor for Planning and Debugging Programs*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 386 (Logo Memo 33), December 1976c.

[Newell 1966]
Newell, Allen, *On the Analysis of Human Problem Solving Protocols*, Carnegie Institute of Technology, Preprint of paper presented at the International Symposium on Mathematical and Computational Methods in the Social Sciences, Rome, June 1966.

[Newell & Simon 1972]
Newell, Allen, and Herbert A. Simon, *Human Problem Solving*, Englewood Cliffs, New Jersey, Prentice Hall, 1972.

[Papert 1971]
Papert, Seymour A., *Teaching Children to be Mathematicians Versus Teaching About Mathematics*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 249, 1971.

[Paxton & Robinson 1975]
Paxton, William and Ann Robinson, "System Integration and Control in a Speech Understanding System," in *American Journal of Computational Linguistics*,

Volume 5, 1975, pp. 5-18.

[Rubin 1975]
Rubin, Andee, *Hypothesis Formation and Evaluation in Medical Diagnosis*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Technical Report 316, January 1975.

[Sacerdoti 1975]
Sacerdoti, Earl, "The Nonlinear Nature of Plans," in *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, Georgia, U.S.S.R., September 1975, pp. 206-218.

[Solomon 1976]
Solomon, Cynthia, "Leading a Child to a Computer Culture," in R. Colman and P. Lorton Jr. (eds.), *Computer Science and Education* (Advance Proceedings of the Association for Computing Machinery Special Interest Groups on Computer Science Education and Computer Uses in Education Joint Symposium, Anaheim, California), SIGCSE Bulletin, Volume 8, Number 1 (SIGCUE Topics Volume 2), February 1976, pp. 79-83.

[Stansfield et al. 1976]
Stansfield, James L., Brian P. Carr, and Ira P. Goldstein, *Wumpus Advisor I: A First Implementation of a Program that Tutors Logical and Probabilistic Reasoning Skills*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 381, October 14, 1976.

[Sussman 1973]
Sussman, Gerald Jay, *A Computational Model of Skill Acquisition*, New York, American Elsevier, 1975; and Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Technical Report 297, 1973.

[Teitelman 1970]
Teitelman, Warren, "Toward a Programming Laboratory," in J.N. Buxton and B. Randell (eds.), *Software Engineering Techniques* (Report on a Conference Sponsored by the NATO Science Committee, Rome Italy, 27-31 October 1969), April 1970, pp. 137-149.

[Teitelman 1974]
Teitelman, Warren, *Interlisp Reference Manual*, Xerox Palo Alto Research Center, 1974.

[Waterman & Newell 1972]
Waterman, D.A., and A. Newell, *Preliminary Results with a System For Automatic Protocol Analysis*, Carnegie-Mellon University, C.I.P. Working Paper 211, May 1972.

[Waterman & Newell 1973]
Waterman, D.A., and A. Newell, "PAS-II: An Interactive Task-Free Version of An Automatic Protocol Analysis System," in *Advance Papers of the Third International Joint Conference on Artificial Intelligence*, Stanford, California, 20-23 August 1973, pp. 431-445.

[Winston 1970]
Winston, Patrick H., *Learning Structural Descriptions From Examples* Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Technical Report 231, September 1970.

[Woods 1970]
Woods, William A., "Transition Network Grammars for Natural Language Analysis," *Communications of the Association For Computing Machinery*, Volume 13, Number 10, October 1970, pp. 591-606.